Fig. 1

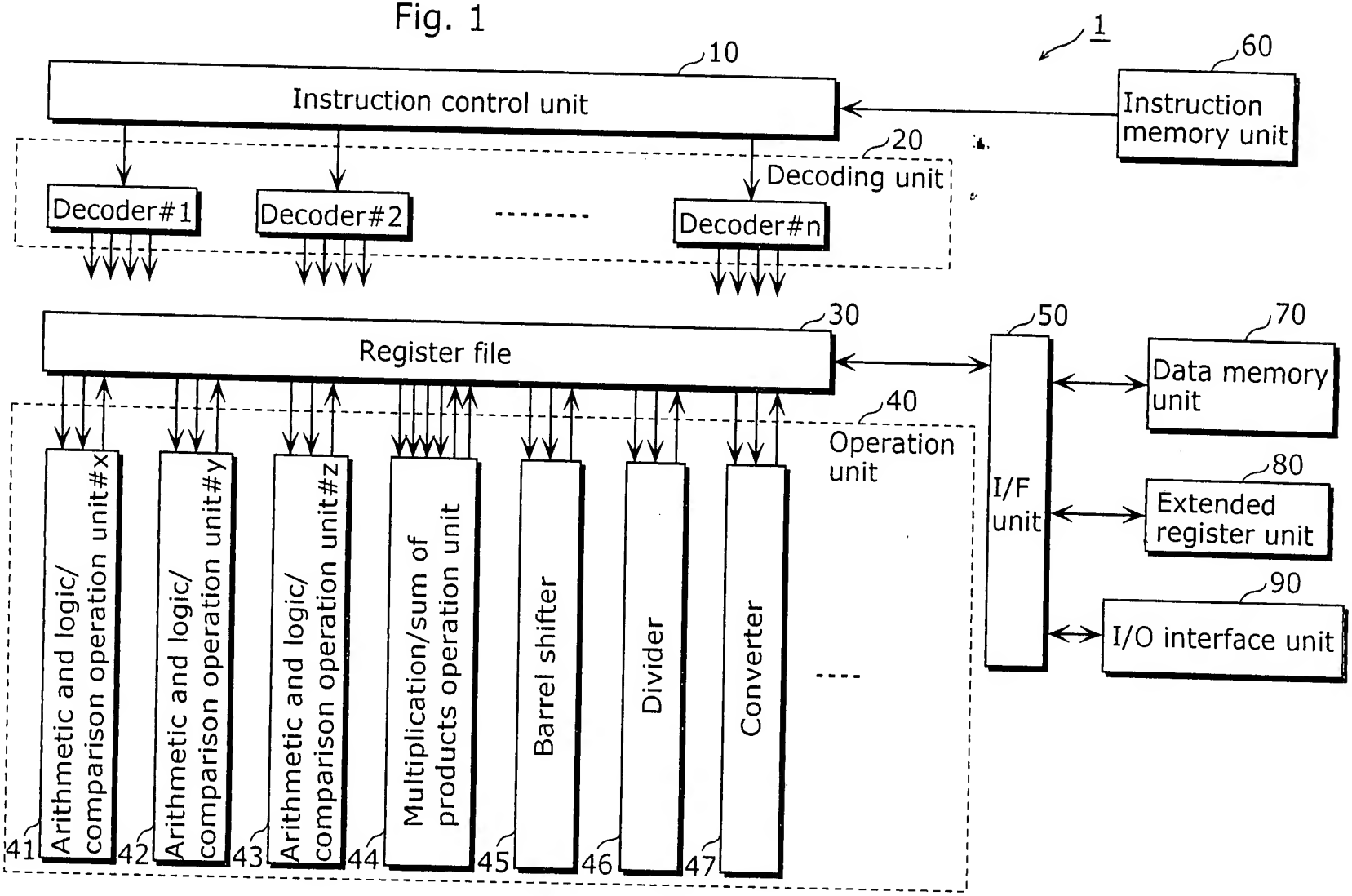

1
10
Instruction control unit
60
Instruction memory unit
20
Decoding unit
Decoder#1
Decoder#2
Decoder#n
30
Register file
50
70
Data memory unit
40
Operation unit
I/F unit
80
Extended register unit
90
I/O interface unit
Arithmetic and logic/ comparison operation unit#x
Arithmetic and logic/ comparison operation unit#y
Arithmetic and logic/ comparison operation unit#z
Multiplication/sum of products operation unit
Barrel shifter
Divider
Converter
41
42
43
44
45
46
47

Fig. 2

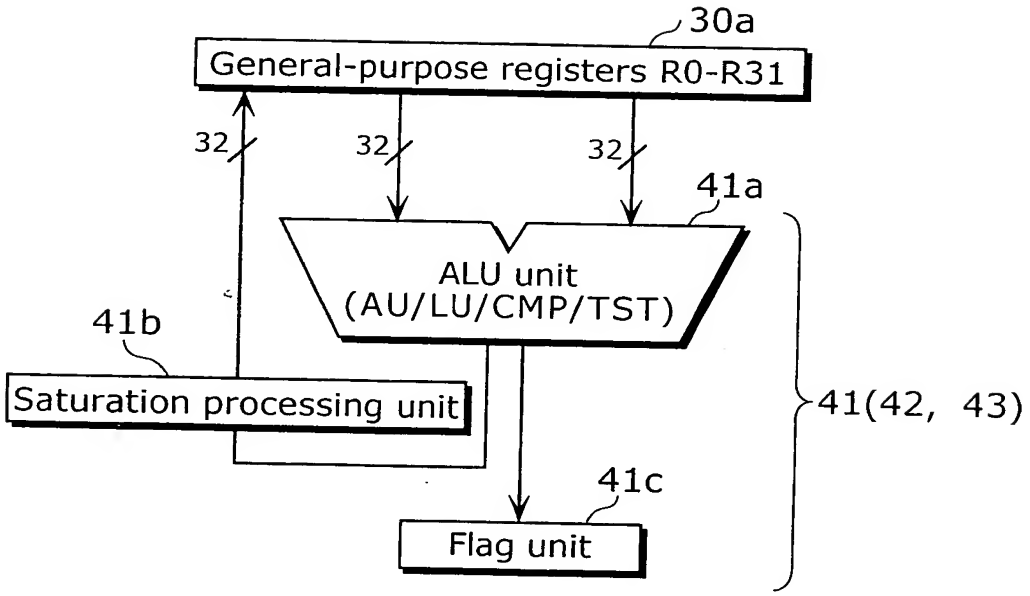

30a
General-purpose registers R0-R31
32
32
32
41a
ALU unit
(AU/LU/CMP/TST)
41b
Saturation processing unit
41(42, 43)
41c
Flag unit

Fig. 3

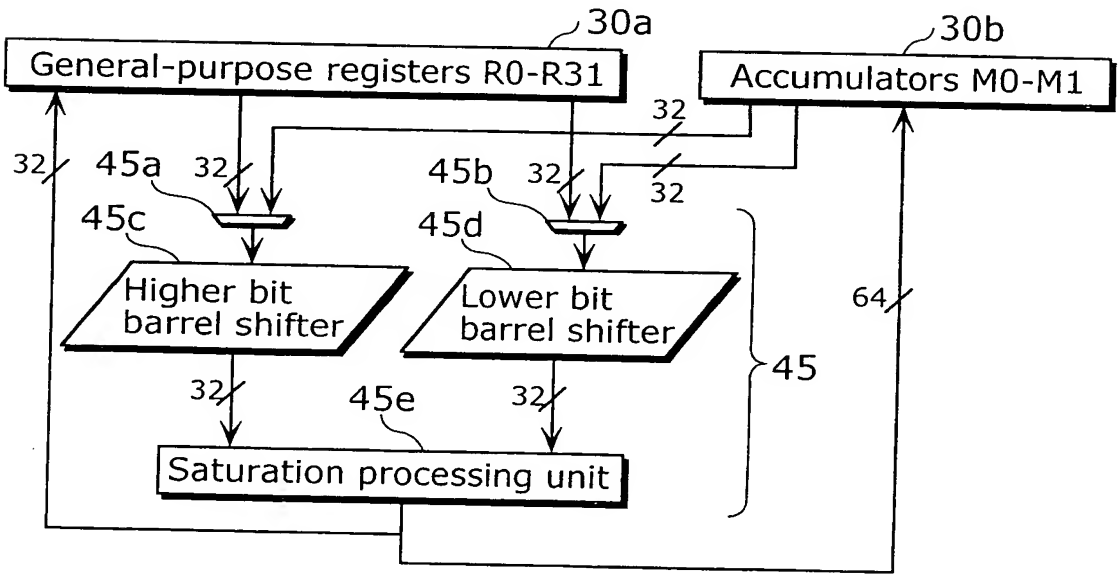

30a
General-purpose registers R0-R31
30b
Accumulators M0-M1
32
32
45a
32
45b
32
32
45c
45d
Higher bit
barrel shifter
Lower bit
barrel shifter
64
45
32
45e
32
Saturation processing unit

Fig. 4

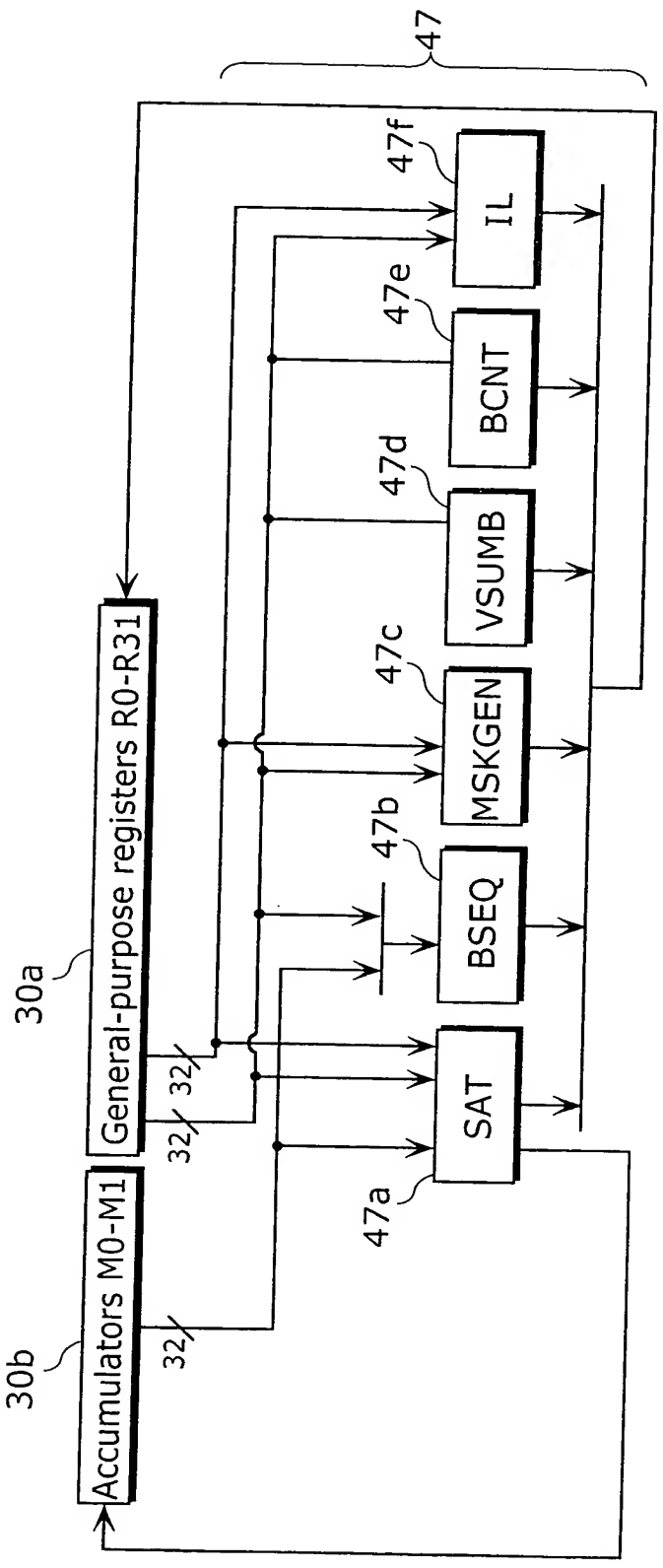

47
47f
IL
47e
BCNT
47d
VSUMB
47c
MSKGEN
47b
BSEQ
47a
SAT
30a
General-purpose registers R0-R31
30b
Accumulators M0-M1
32
32
32

Fig. 5

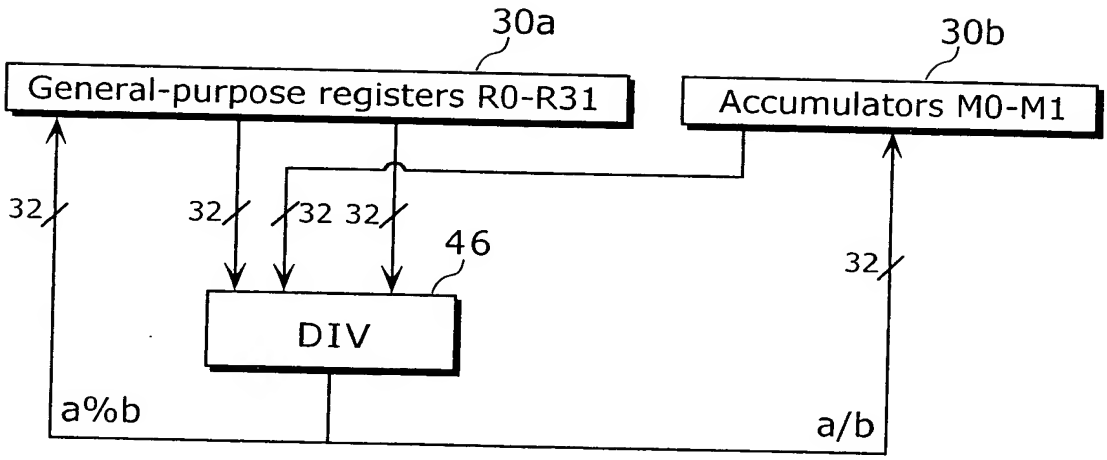

30a
30b
General-purpose registers R0-R31
Accumulators M0-M1
32
32
32 32
46
32
DIV
a%b
a/b

Fig. 6

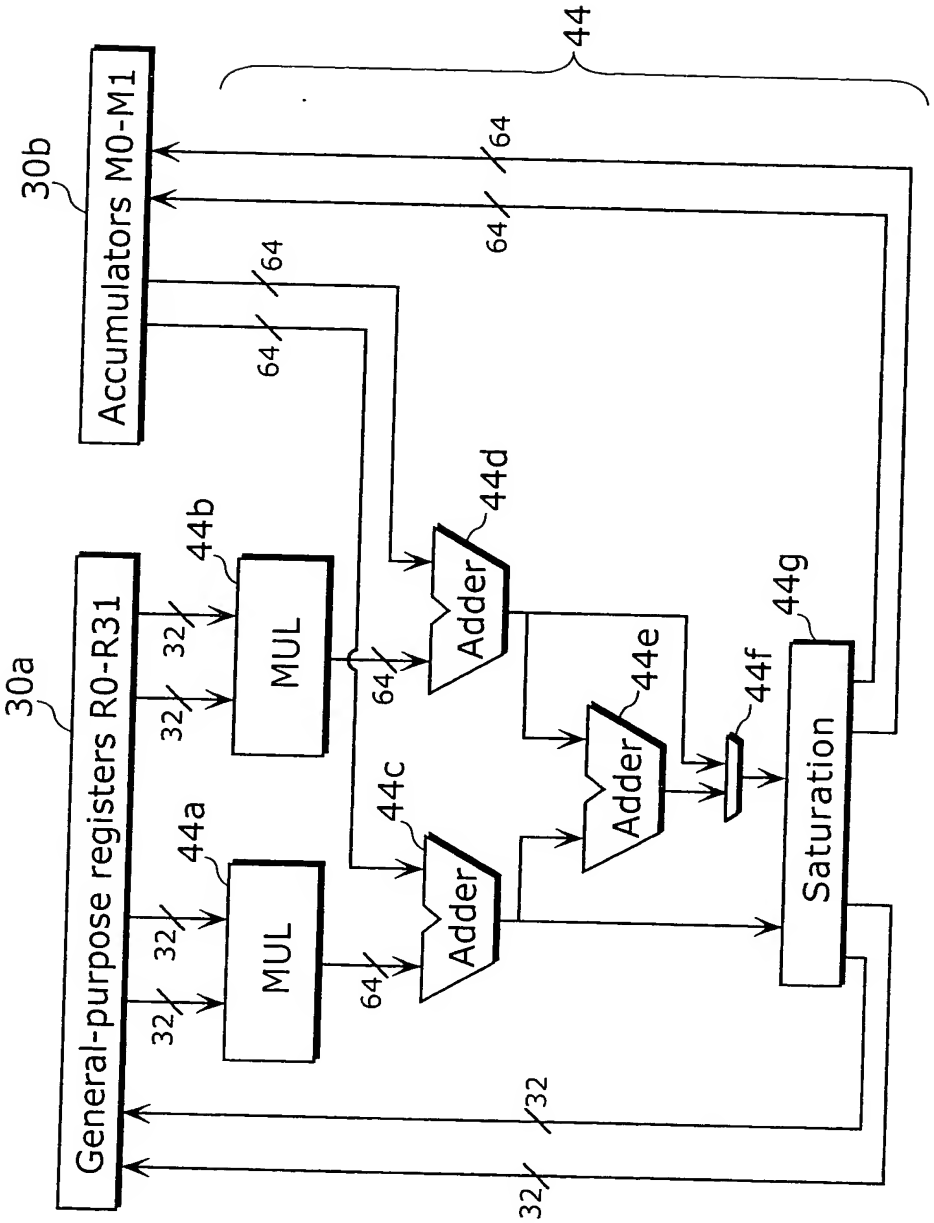

44
30b
Accumulators M0-M1
64
64
64
64
30a
General-purpose registers R0-R31
44b
MUL
32
32
44a
MUL
32
32
64
64
44d
Adder
44c
Adder
44e
Adder
44f
44g
Saturation
32
32

Fig. 7

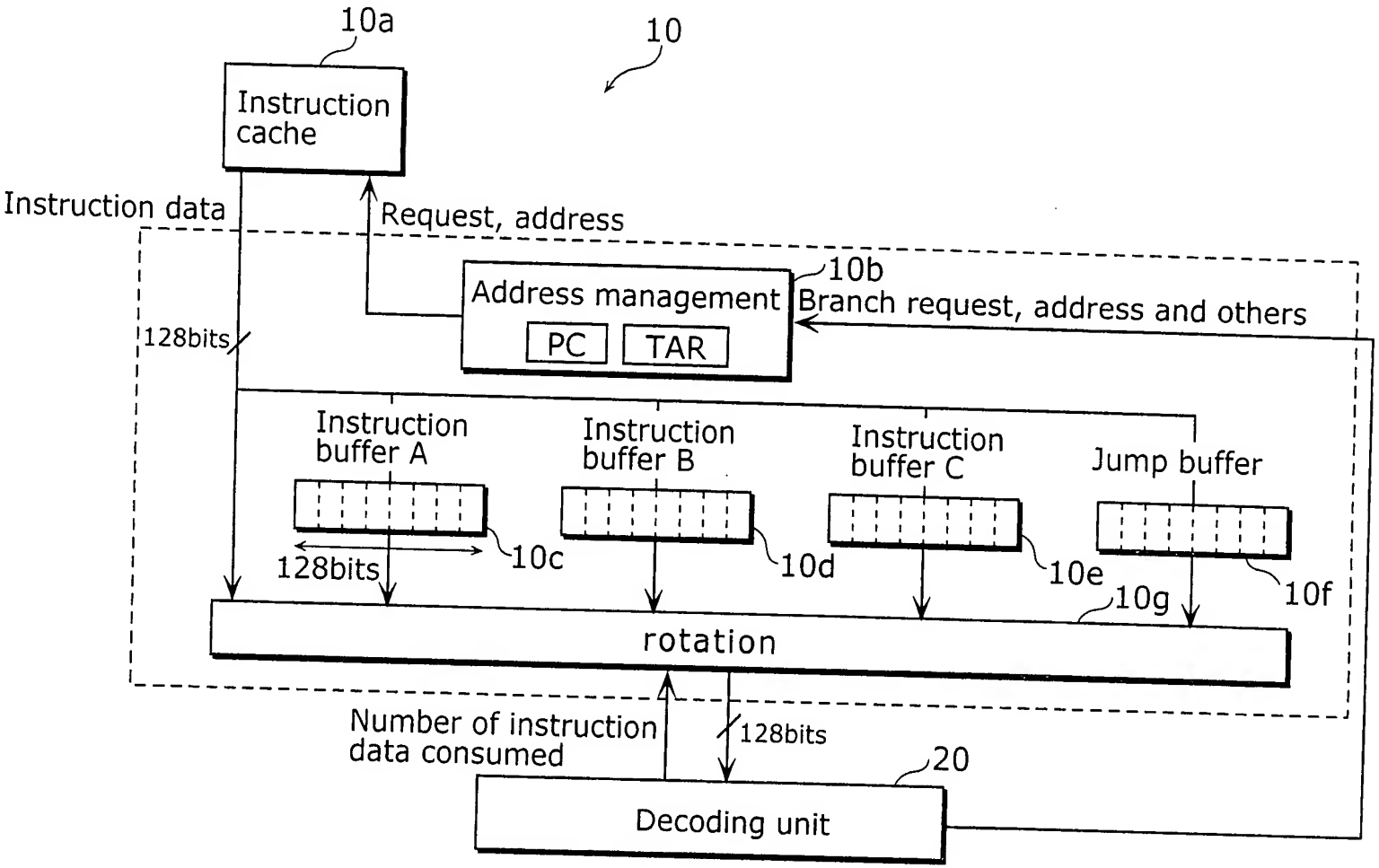

10a
10
Instruction cache
Instruction data
Request, address
10b
Address management
PC
TAR
Branch request, address and others
128bits
Instruction buffer A
Instruction buffer B
Instruction buffer C
Jump buffer
128bits
10c
10d
10e
10f
10g
rotation
Number of instruction data consumed
128bits
20
Decoding unit

Fig. 8

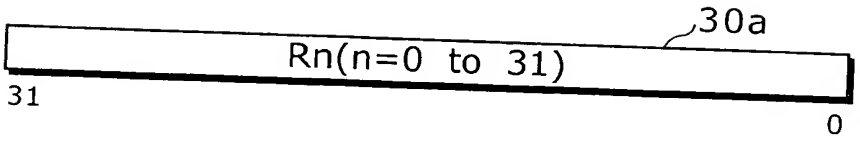


Fig. 9

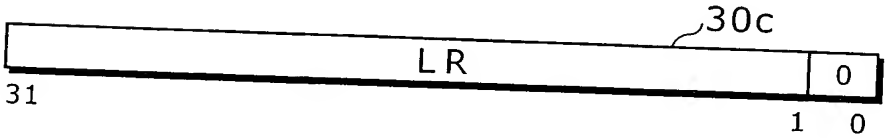


Fig. 10

Fig. 11

Bit	31	30	29	28	27	26	25	24
Bit name	reserved	SWE	FXP	reserved	IH	EH	PL	
Bit	23	22	21	20	19	18	17	16
Bit name	FIE3	FIE2	FIE1	FIE0	reserved	reserved	AEE	IE
Bit	15	14	13	12	11	10	9	8
Bit name	Reserved							
Bit	7	6	5	4	3	2	1	0
Bit name	IM[7:0]							

Fig. 12

32

Bit	31	30	29	28	27	26	25	24
Bit name	ALN		reserved	BPO				
Bit	23	22	21	20	19	18	17	16
Bit name	reserved				VC3	VC2	VC1	VC0
Bit	15	14	13	12	11	10	9	8
Bit name	reserved						OVS	CAS
Bit	7	6	5	4	3	2	1	0
Bit name	C7	C6	C5	C4	C3	C2	C1	C0

Fig. 13A

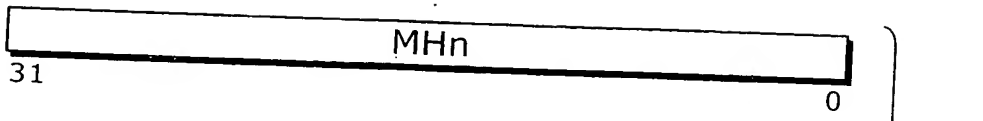
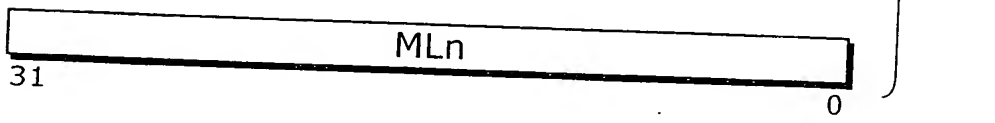


Fig. 13B



30b

Fig. 14

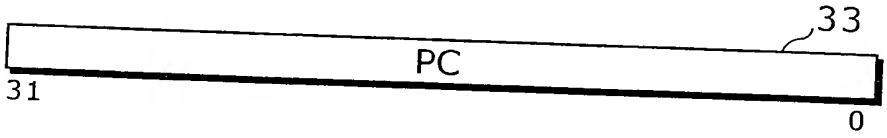


Fig. 15

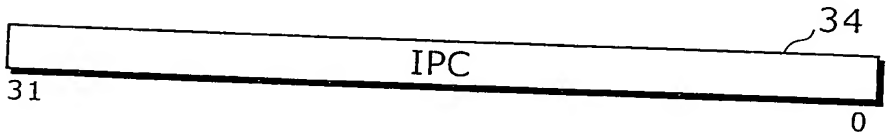


Fig. 16

Fig. 17

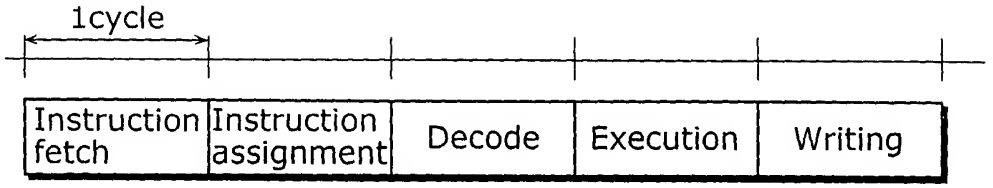

1cycle
Instruction fetch
Instruction assignment
Decode
Execution
Writing

Fig. 18

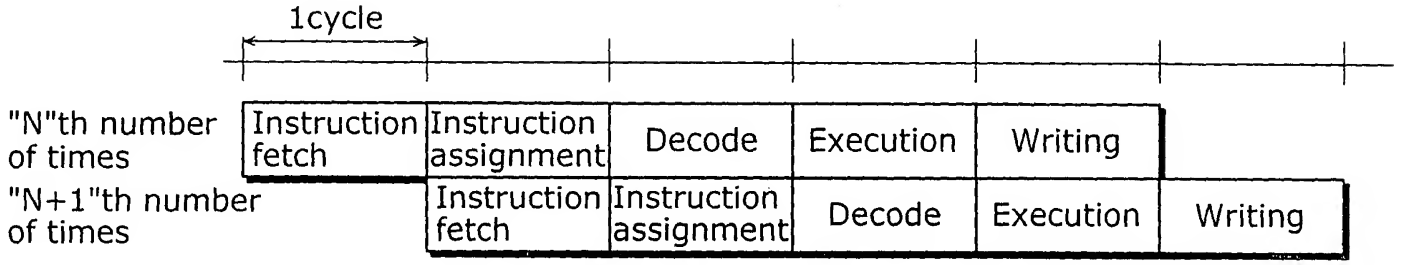

1cycle
"N"th number of times
"N+1"th number of times
Instruction fetch
Instruction assignment
Decode
Execution
Writing
Instruction fetch
Instruction assignment
Decode
Execution
Writing

Fig. 19

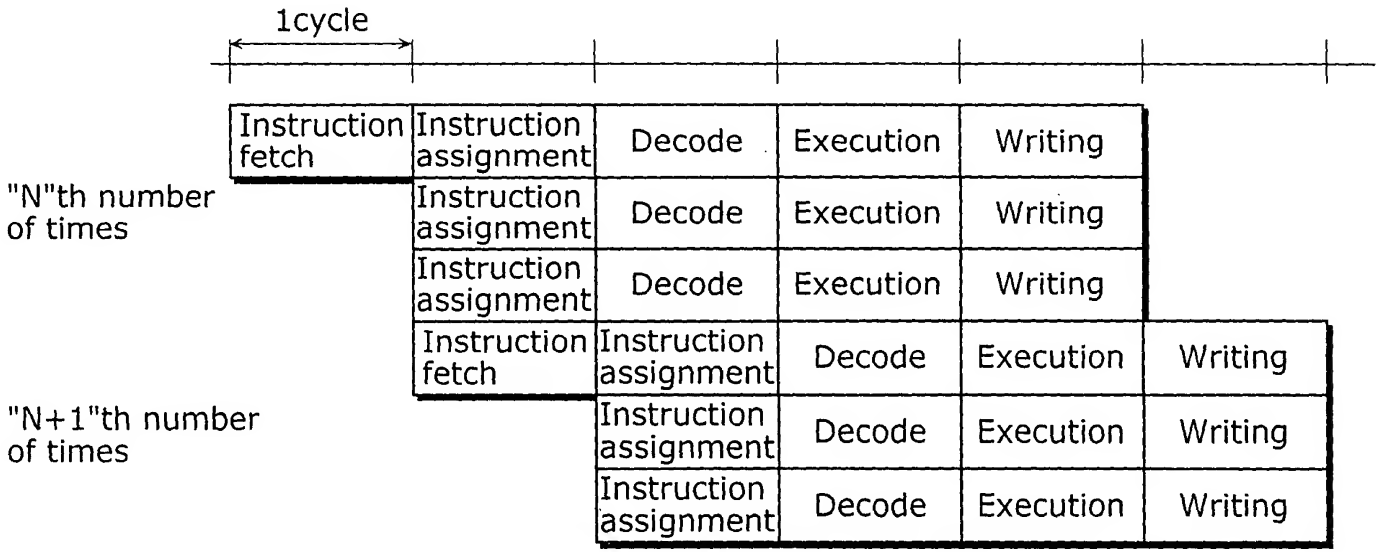

1cycle
Instruction fetch
Instruction assignment
Decode
Execution
Writing
"N"th number of times
"N+1"th number of times

Fig. 20

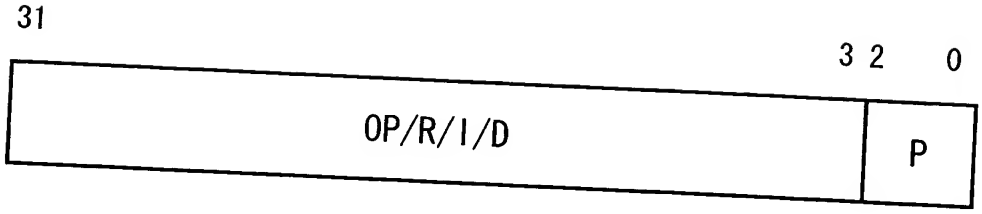


Fig. 21

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
ALU add system	SINGLE	Word	add	Rc,Ra,Rb					32
				Rb,Ra,i12s					
				SP,i19s					
				Ra2,Rb2					16
				Rc3,Ra3,Rb3					
				Ra2,i05s					
				SP,i11s					
			addu	Rb,GP,i16u					32
				Rb,SP,i16u					
				Ra3,SP,i08u					16
			addc	Rc,Ra,Rb	W:cas,c0:c1		Addition with carry		
			addvw	Rc,Ra,Rb	W:ovs		Addition with overflow		
			adds	Rc,Ra,Rb			Ra + Rb → >>1 Rc		32
			addsr	Rc,Ra,Rb			Ra + Rb + 1 → >>1 Rc		
			s1add	Rc,Ra,Rb					
				Rc3,Ra3,Rb3			Ra + >>1 Rb (>>2) → Rc		16
			s2add	Rc,Ra,Rb					32
				Rc3,Ra3,Rb3					16
			addmsk	Rc,Ra,Rb	R:BP0		Ra, Rb, CFR.BP → Rc		
			addarvw	Rc,Ra,Rb					
		Half word	faddvh	Rc,Ra,Rb	W:ovs				
	SIMD		vaddh	Rc,Ra,Rb					
			vaddhvh	Rc,Ra,Rb	W:ovs				
			vsaddh	Rb,Ra,i08s			+Immediate value		
			vaddsh	Rc,Ra,Rb			(+1) (+1) → >>1 (With rounding)		
			vaddsrh	Rc,Ra,Rb					
		Half word	vaddhvc	Rc,Ra,Rb	R:VC			A	32
			vaddrhvc	Rc,Ra,Rb					
			vxaddh	Rc,Ra,Rb					
			vxaddhvh	Rc,Ra,Rb	W:ovs				
			vhaddh	Rc,Ra,Rb					
			vhaddhvh	Rc,Ra,Rb	W:ovs				
			vladdh	Rc,Ra,Rb					
			vladdhvh	Rc,Ra,Rb	W:ovs				
		Byte	vaddb	Rc,Ra,Rb					
			vsaddb	Rb,Ra,i08s			(Immediate value)		
			vaddsb	Rc,Ra,Rb			(+1)(+1)(+1)(+1) → >>1 (With rounding)		
			vaddsrb	Rc,Ra,Rb					

Fig. 22

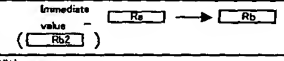
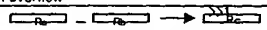
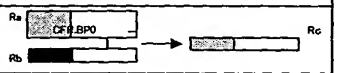
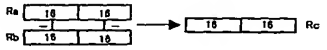
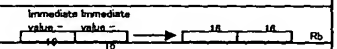
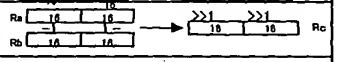
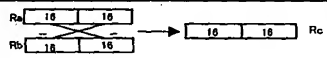
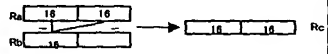
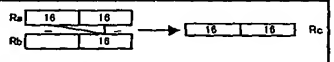
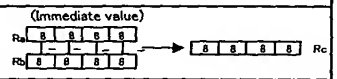
Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 / 16
ALU sub system	SINGLE	Word	sub	Rc,Rb,Ra Rb2,Ra2 Rc3,Rb3,Ra3				A	32 16
			rsub	Rb,Ra,i08s Ra2,Rb2 Ra2,i04s					32 16
			subc	Rc,Rb,Ra	W:cas,c0:c1		With carry		
			subvw	Rc,Rb,Ra	W:ovs		With overflow		
			subs	Rc,Rb,Ra					
			submsk	Rc,Rb,Ra	R:BP0				
		Half word	fsubvh	Rc,Rb,Ra					
	SIMD		vsubh	Rc,Rb,Ra					
			vsubhvh	Rc,Rb,Ra	W:ovs				
			vsrsubh	Rb,Ra,i08s					32
			vsubsh	Rc,Rb,Ra					
		Half word	vxsubh	Rc,Rb,Ra					
			vxsubhvh	Rc,Rb,Ra	W:ovs				
			vhsubh	Rc,Rb,Ra					
			vhsubhvh	Rc,Rb,Ra	W:ovs				
			vlsubh	Rc,Rb,Ra					
			vlsubhvh	Rc,Rb,Ra	W:ovs				
		Byte	vsubb	Rc,Rb,Ra					
			vsrsubb	Rb,Ra,i08s					
			vasubb	Rc,Rb,Ra	R:VC				

Fig. 23

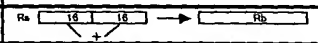
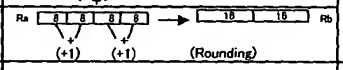
Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 / 16
ALU logic system	SINGLE	Word	and	Rc,Ra,Rb			AND	A	32
				Rb,Ra,i08u					
				Ra2,Rb2					16
			andn	Rc,Ra,Rb					32
				Rb,Ra,i08u					
				Ra2,Rb2					16
			or	Rc,Ra,Rb			OR		32
				Rb,Ra,i08u					
				Ra2,Rb2					16
			xor	Rc,Ra,Rb			Exclusive OR		32
				Rb,Ra,i08u					
				Ra2,Rb2					16
ALU mov system	SINGLE	Word	mov	Rb,Reg32			Reg32 = TAR LR SVR PSR CFR MH0 MH1 ML0 ML1 EPSR IPC IPSR PC EPC PSR0 PSR1 PSR2 PSR3 CFR0 CFR1 CFR2 CFR3	A	32
				Reg32,Rb					
				Rb2,Reg16			Reg16 = TAR LR MH0 MH1		16
				Reg16,Rb2					
				Ra2,Rb					32
				Ra,i16s					
				Ra2,i08s					16
			movp	Rc:Rc+1,Ra,Rb			Rc<-Ra; Rc+1<-Rb;		32
			movcf	Ck,Cj,Cm,Cn			Ci<-Cj; Cm<-Cn;		
			mvclovs	Cm:Cm+1	W:ovs		Cm:Cm+1<-CFR.OVS; CFR.OVS; CFR.OVS<-0;		16
			mvclcas	Cm:Cm+1	W:cas		Cm:Cm+1<-CFR.CAS; CFR.CAS; CFR.CAS<-0;		
			sethi	Ra,i16s					32
ALU max min system	SINGLE	Word	max	Rc,Ra,Rb	W:c0:c1		Rc <- max(Ra,Rb)	A	32
			min	Rc,Ra,Rb	W:c0:c1		Rc <- min(Ra,Rb)		
	SIMD	Half word	vmaxh	Rc,Ra,Rb					
			vminh	Rc,Ra,Rb					
		Byte	vmaxb	Rc,Ra,Rb					
			vminb	Rc,Ra,Rb					
ALU abs system	SINGLE		abs	Rb,Ra			Absolute value	A	32
			absvw	Rb,Ra	W:ovs		With overflow		
			fabsvh	Rb,Ra	W:ovs				
	SIMD	Half word	vabshvh	Rb,Ra	W:ovs				
ALU neg system	SINGLE	Word	negvw	Rb,Ra	W:ovs			A	32
		Half word	fnegvh	Rb,Ra	W:ovs				
	SIMD		vneghvh	Rb,Ra	W:ovs				
ALU sum system	SIMD	Half word	vsumh	Rb,Ra				A	32
			vsumh2	Rb,Ra					
			vsumrh2	Rb,Ra					
		Byte	vabssumb	Rc,Ra,Rb				C	
ALU other	SINGLE		frndvh	Rb,Ra	W:ovs		Rounding	C	32
	SIMD		vfrndvh	Rb,Mn	W:ovs				
			vsel	Rc,Ra,Rb	R:VC			A	
			vsgnh	Rb,Ra				C	

Fig. 24

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 / 16
CMP	SINGLE		cmpCCn	Cm,Ra,Rb,Cn Cm,Ra,i05s,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		CC = eq, ne, gt, ge, gtu, geu, le, lt, leu, leu Cm <- result & Cn; (Cm+1 <- ~result & Cn;)	A	32
			cmpCCa	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		Cm <- result & Cn; Cm+1 <- ~(result & Cn);		
			cmpCCo	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		Cm <- result | Cn; Cm+1 <- ~(result | Cn);		
			cmpCC	C6,Ra2,Rb2 C6,Ra2,i04s	W:CF		CC = eq, ne, gt, ge, le, lt C6 <- result		16
			tstzn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) & Cn; (Cm+1 <- ~(Ra & Rb ==0) & Cn;)		32
			tstza	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) & Cn; Cm+1 <- ~((Ra & Rb == 0) & Cn);		
			tstzo	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) | Cn; Cm+1 <- ~((Ra & Rb == 0) | Cn);		
			tstnn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) & Cn; (Cm+1 <- ~(Ra & Rb != 0) & Cn;)		
			tstna	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) & Cn; Cm+1 <- ~((Ra & Rb != 0) & Cn);		
			tstno	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) | Cn; Cm+1 <- ~((Ra & Rb != 0) | Cn);		
			tstz	C6,Ra2,Rb2 C6,Ra2,i04u	W:CF		C6 <- (Ra2&Rb2 == 0)		16
			tstn	C6,Ra2,Rb2 C6,Ra2,i04u	W:CF		C6 <- (Ra2&Rb2 != 0)		
	SIMD	Half word	vcmpCCh vscmpCCh	Ra,Rb Ra,i05s	W:CF		CC = eq, ne, gt, le, ge, lt		32
		Byte	vcmpCCb vscmpCCb	Ra,Rb Ra,i05s	W:CF		CC = eq, ne, gt, le ge, lt		

Fig. 25

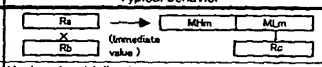
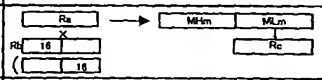
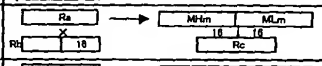
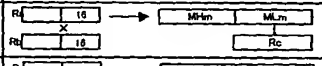
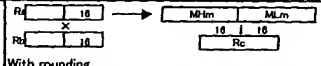
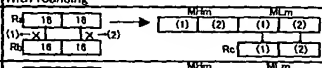
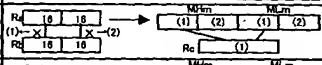
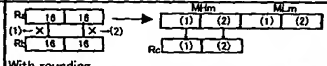
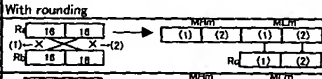
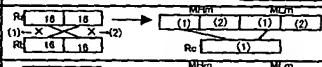
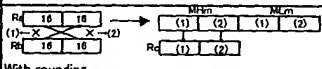
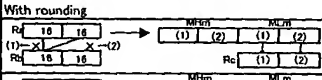
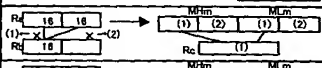
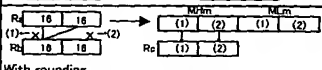
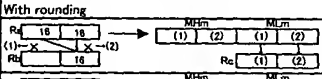
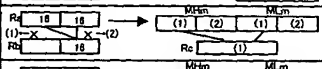
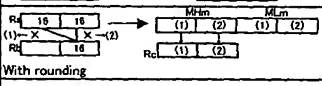
Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
mul system	SINGLE	Word × Word	mul	Mm,Rc,Ra,Rb Mm,Rb,Ra,i08s				X2	32
			mulu	Mm,Rc,Ra,Rb Mm,Rb,Ra,i08s			Unsigned multiplication		
			fmulww	Mm,Rc,Ra,Rb		fxp	Fixed point operation		
		Word × Half word	hmul	Mm,Rc,Ra,Rb				X1	
			lmul	Mm,Rc,Ra,Rb					
			fmulhww	Mm,Rc,Ra,Rb		fxp			
		Half word × Half word	fmulhw	Mm,Rc,Ra,Rb		fxp			
			fmulhh	Mm,Rc,Ra,Rb		fxp			
			fmulhhr	Mm,Rc,Ra,Rb		fxp	With rounding		
	SIMD	HALF WORD × HALF WORD	vmul	Mm,Rc,Ra,Rb				X2	
			vfmulw	Mm,Rc,Ra,Rb		fxp			
			vfmulh	Mm,Rc,Ra,Rb		fxp			
			vfmulhr	Mm,Rc,Ra,Rb		fxp	With rounding		
			vxmul	Mm,Rc,Ra,Rb		fxp			
			vxfmulw	Mm,Rc,Ra,Rb		fxp			
			vxfmulh	Mm,Rc,Ra,Rb		fxp			
			vxfmulhr	Mm,Rc,Ra,Rb		fxp	With rounding		
			vhmul	Mm,Rc,Ra,Rb		fxp			
			vhfmulw	Mm,Rc,Ra,Rb		fxp			
			vhfmulh	Mm,Rc,Ra,Rb		fxp			
			vhfmulhr	Mm,Rc,Ra,Rb		fxp	With rounding		
			vlmul	Mm,Rc,Ra,Rb		fxp			
			vlfmulw	Mm,Rc,Ra,Rb		fxp			
			vlfmulh	Mm,Rc,Ra,Rb		fxp			
			vlfmulhr	Mm,Rc,Ra,Rb		fxp	With rounding		
		Word × Half word	vpfmulhww	Mm,Rc:Rc+1,Ra,Rb Mm,Rc:Rc+1,Ra,Rb		fxp fxp			

Fig. 26

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 / 16
mac system	SINGLE	Word × Word	mac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using mul	X2	32
			fmacww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using fmulww		
		Word × Half word	hmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using hmul	X1	16
			lmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using lmul		32
			fmachww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using fmulhww		16
		Half word × Half word	fmachw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using fmulhw		32
			fmachh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using fmulhh		
			fmachhr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	With rounding		
	SIMD	HALF WORD × HALF WORD	vmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using vmul	X2	16
			vfmacw	Mm,Rc,Ra,Rb,Mn		fxp	Sum of products operation using vfmulw		32
			vxmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using vxmul		16
			vxfmacw	Mm,Rc,Ra,Rb,Mn		fxp	Sum of products operation using vxfmulw		32
			vxfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using vxfmulh		
			vxfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	With rounding		
			vhmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using vhmul		
			vhfmacw	Mm,Rc,Ra,Rb,Mn		fxp	Sum of products operation using vhfmulw		
			vhfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using vhfmulh		
			vhfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx					
		WORD × HALF WORD	vlmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using vlmul		32
			vlfmacw	Mm,Rc,Ra,Rb,Mn		fxp	Sum of products operation using vlfmulw		
			vfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using vfmulh		
			vfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	With rounding		
			vlfmach	Mm,Rc,Ra,Rb,Mn Mm,Rc,Ra,Rb,Mn		fxp	Sum of products operation using vlfmulh		
			vlfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	With rounding		
		Word × Half word	vpfmachww	Mm,Rc:Rc+1,Ra,Rb,Mn		fxp			

Fig. 27

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
msu system	SINGLE	Word × Word	msu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using mul	X2	32
			fmsuww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using fmulww		
		Word × Half word	hmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using hmul	X1	
			lmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using lmul		
			fmsuhww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using fmulhww		
		Half word × Half word	fmsuhw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using fmulhw		
			fmsuhh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using fmulhh		
			fmsuhhr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	With rounding		
		HALF WORD × HALF WORD	vmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using vmul	X2	
			vfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	Difference of products operation using vfmul		
			vfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using vfmulh		
			vxmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using vxmul		
			vxfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	Difference of products operation using vxfmulw		
			vxfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using vxfmulh		
			vhmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using vhmul		
			vhfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	Difference of products operation using vhfmulw		
			vhfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using vhfmulh		
			vlmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using vlmul		
			vlfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	Difference of products operation using vlfmulw		
			vlfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using vlfmulh		

Fig. 28

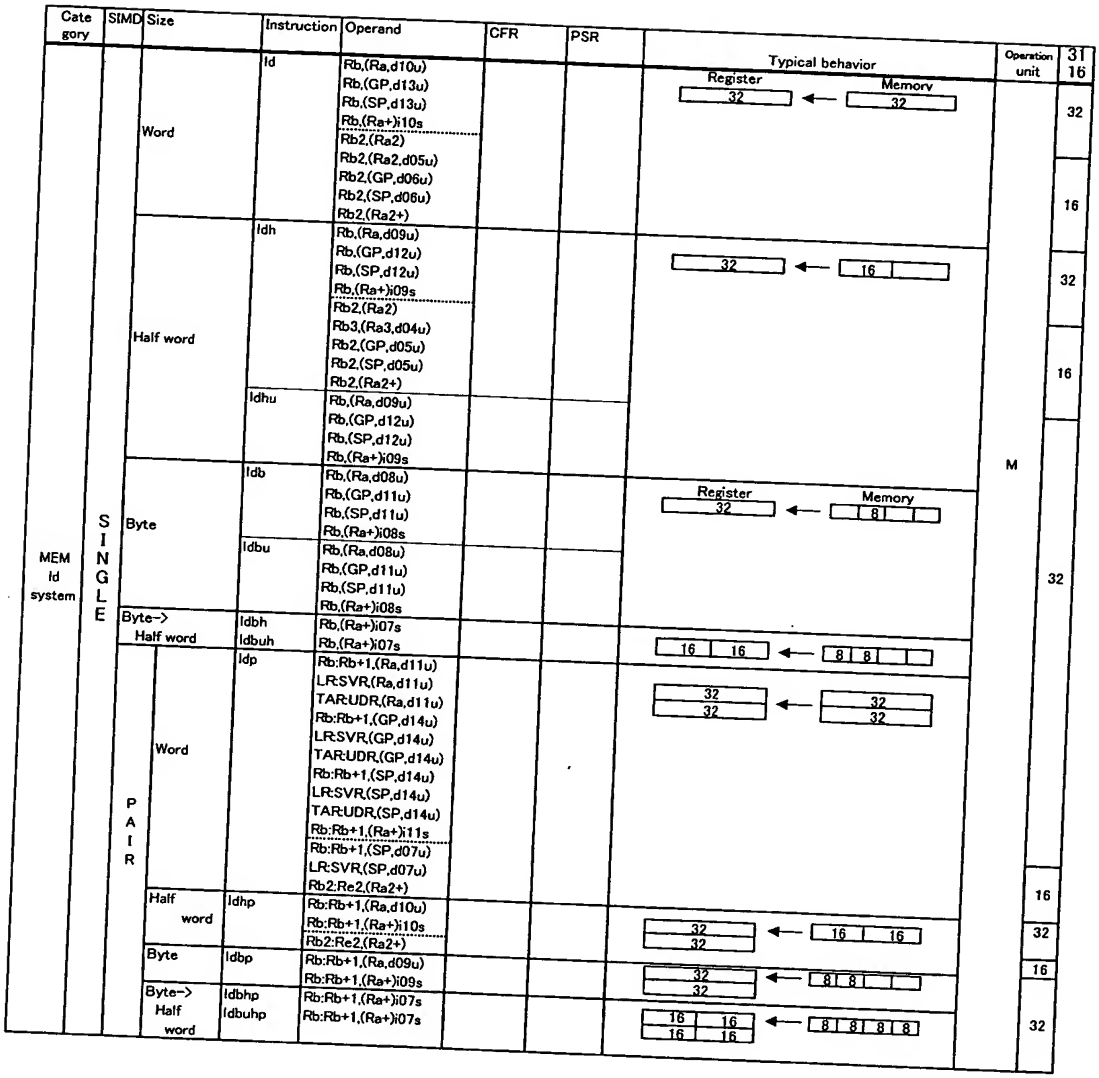
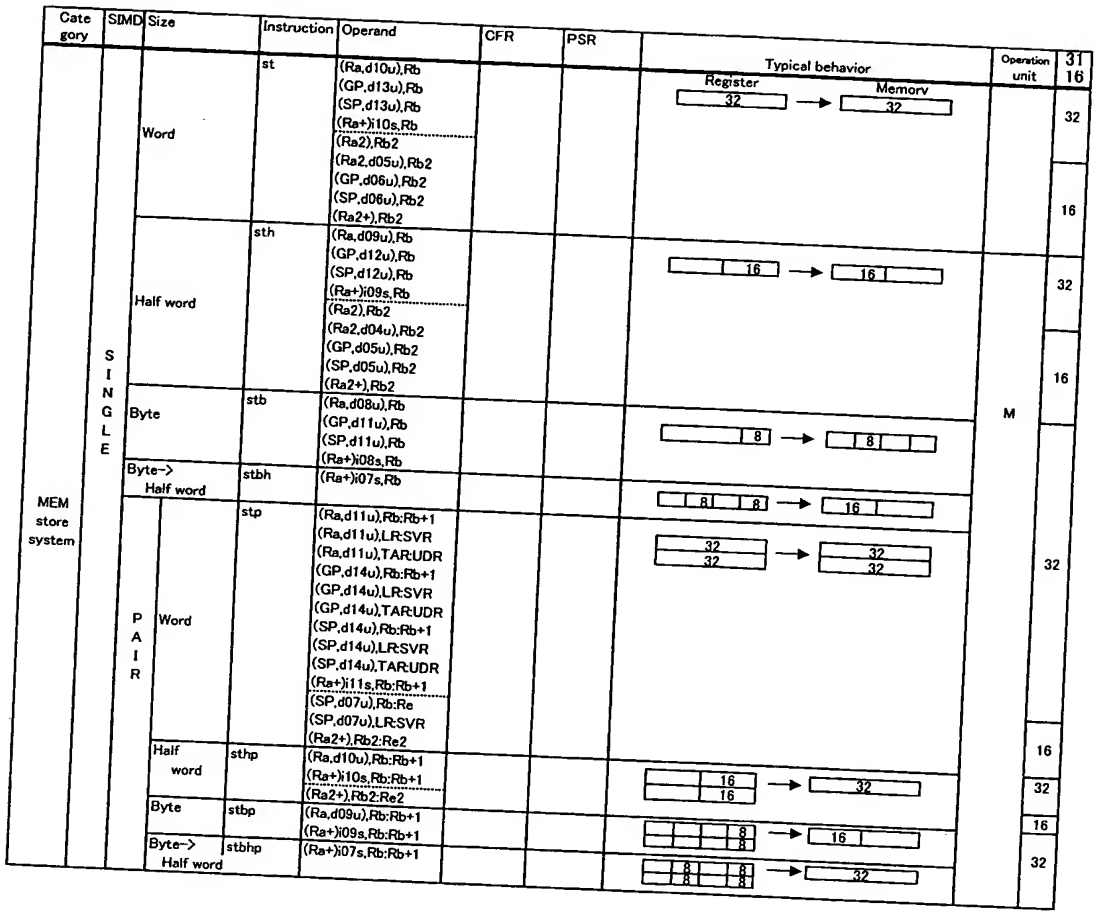

Cate gory
SIMD
Size
Instruction
Operand
CFR
PSR
Typical behavior
Operation unit
31
16
MEM ld system
SINGLE
PAIR
Word
Half word
Byte
Byte-> Half word
ld
ldh
ldhu
ldb
ldbu
ldbh
ldbuh
ldp
ldhp
ldbp
ldbhp
ldbuhp
Rb,(Ra,d10u)
Rb,(GP,d13u)
Rb,(SP,d13u)
Rb,(Ra+)i10s
Rb2,(Ra2)
Rb2,(Ra2,d05u)
Rb2,(GP,d06u)
Rb2,(SP,d06u)
Rb2,(Ra2+)
Rb,(Ra,d09u)
Rb,(GP,d12u)
Rb,(SP,d12u)
Rb,(Ra+)i09s
Rb3,(Ra3,d04u)
Rb2,(GP,d05u)
Rb2,(SP,d05u)
Rb,(Ra,d08u)
Rb,(GP,d11u)
Rb,(SP,d11u)
Rb,(Ra+)i08s
Rb,(Ra+)i07s
Rb:Rb+1,(Ra,d11u)
LR:SVR,(Ra,d11u)
TAR:UDR,(Ra,d11u)
Rb:Rb+1,(GP,d14u)
LR:SVR,(GP,d14u)
TAR:UDR,(GP,d14u)
Rb:Rb+1,(SP,d14u)
LR:SVR,(SP,d14u)
TAR:UDR,(SP,d14u)
Rb:Rb+1,(Ra+)i11s
Rb:Rb+1,(SP,d07u)
LR:SVR,(SP,d07u)
Rb2:Re2,(Ra2+)
Rb:Rb+1,(Ra,d10u)
Rb:Rb+1,(Ra+)i10s
Rb2:Re2,(Ra2+)
Rb:Rb+1,(Ra,d09u)
Rb:Rb+1,(Ra+)i09s
Rb:Rb+1,(Ra+)i07s
Rb:Rb+1,(Ra+)i07s
Register
Memory
M

Fig. 29



Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
MEM store system	SINGLE	Word	st	(Ra,d10u),Rb (GP,d13u),Rb (SP,d13u),Rb (Ra+)i10s,Rb			Register 32 → Memory 32		32
				(Ra2),Rb2 (Ra2,d05u),Rb2 (GP,d06u),Rb2 (SP,d06u),Rb2 (Ra2+),Rb2					16
		Half word	sth	(Ra,d09u),Rb (GP,d12u),Rb (SP,d12u),Rb (Ra+)i09s,Rb			16 → 16	M	32
				(Ra2),Rb2 (Ra2,d04u),Rb2 (GP,d05u),Rb2 (SP,d05u),Rb2 (Ra2+),Rb2					16
		Byte	stb	(Ra,d08u),Rb (GP,d11u),Rb (SP,d11u),Rb (Ra+)i08s,Rb			8 → 8		32
		Byte-> Half word	stbh	(Ra+)i07s,Rb			8 8 → 16		
	PAIR	Word	stp	(Ra,d11u),Rb:Rb+1 (Ra,d11u),LR:SVR (Ra,d11u),TAR:UDR (GP,d14u),Rb:Rb+1 (GP,d14u),LR:SVR (GP,d14u),TAR:UDR (SP,d14u),Rb:Rb+1 (SP,d14u),LR:SVR (SP,d14u),TAR:UDR (Ra+)i11s,Rb:Rb+1			32 32 → 32 32		
				(SP,d07u),Rb:Re (SP,d07u),LR:SVR (Ra2+),Rb2:Re2					16
		Half word	sthp	(Ra,d10u),Rb:Rb+1 (Ra+)i10s,Rb:Rb+1			16 16 → 32		32
				(Ra2+),Rb2:Re2					16
		Byte	stbp	(Ra,d09u),Rb:Rb+1 (Ra+)i09s,Rb:Rb+1			8 8 → 16		32
		Byte-> Half word	stbhp	(Ra+)i07s,Rb:Rb+1			8 8 8 8 → 32		

Fig. 30

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
BRA			setlr	d09s			Set LR	B	32
				C5,d09s			Store instruction fetched from LR in branch buffer		
			settar	d09s			Set TAR		
				C6,d09s	W:c6		Store instruction fetched from TAR in branch buffer		
				C6,C2:C4,d09s	W:c2:c4,c6				
				C6,Cm,d09s	W:c6,cm				
				C6,C4,d09s	W:c6				
			setbb	LR			Store instruction fetched from LR in branch buffer		16
				TAR			Store instruction fetched from TAE in branch buffer		
			jloop	C5,LR,Ra,i08s	W:c5		Only predicate [c5]		32
				C6,TAR,Ra,i08s	W:c6		Only predicate [c6]		
				C6,C2:C4,TAR,Ra,i08s	W:c2:c4,c6				
				C6,Cm,TAR,Ra,i08s	W:c6,cm				
				C6,TAR,Ra2	W:c6				16
				C6,C2:C4,TAR,Ra2	W:c2:c4,c6				
				C6,Cm,TAR,Ra2	W:c6				
			jmp	TAR					
				LR					
			jmpl	TAR	R:CF				
				LR					
			jmpf	TAR					
				LR					
				Cm,TAR					
				C6,C2:C4,TAR					
			jmpr	LR					
			br	d20s					32
				d09s			Only predicates [c6][c7]		16
			brl	d20s	R:CF				32
				d09s					16
			rti			W:PSR			
						R:eh			

Fig. 31

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 / 16
BS asl system	SINGLE	Word	asl	Rc,Ra,Rb Rb,Ra,i05u Ra2,i04u			Left shift <Rb[0:4] Ra → Rc	S1	32 16
			faslvw	Rc,Ra,Rb Rb,Ra,i05u Rc,Ra,Rb Rb,Ra,i05u	W:ovs		<Rb[0:4] With saturation Ra → Rc		
		Pair word	aslp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			<Ra[0:4] MHm | MLm → MHm | MLm Rb	S2	
			faslpvw	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u	W:ovs		<Rb[0:4] With saturation MHm | MLm → MHm | MLm Rb		
	SIMD	Word	vasl	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u			MHm | MLm → MHm | MLm (1)<Ra[0:4] (2)<Ra[0:4] Rb	S2	32
			vfaslvw	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u	W:ovs		MHm | MLm → MHm | MLm (1)<<Ra[0:4] (2)<<Ra[0:4] Rb		
		Half word	vaslh	Rc,Ra,Rb Rb,Ra,i04u			<Rb[0:3] <Rb[0:3] Ra (1) (2) → (1) (2) Rc	S1	
			vfaslvh	Rc,Ra,Rb Rb,Ra,i04u	W:ovs		<Rb[0:3] <Rb[0:3] Ra (1) (2) → (1) (2) Rc With saturation		
		Byte	vaslb	Rc,Ra,Rb Rb,Ra,i03u			<Rb[0:2] <Rb[0:2] <Rb[0:2] <Rb[0:2] Ra (1) (2) (3) (4) → (1) (2) (3) (4) Rc		
BS asr system	SINGLE	Word	asr	Rc,Ra,Rb Rb,Ra,i05u Ra2,i04u			Arithmetic shift right >>Rb[0:4] Ra → Rc	S1	32 16
		Pair word	asrp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			>>Ra[0:4] MHm | MLm → MHm | MLm Rb	S2	
	SIMD	Word	vasr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u			MHm | MLm → MHm | MLm (1)>>Ra[0:4] (2)>>Ra[0:4] Rb		32
		Half word	vasrh	Rc,Ra,Rb Rb,Ra,i04u			>>Rb[0:3] >>Rb[0:3] Ra (1) (2) → (1) (2) Rc	S1	
		Byte	vasrb	Rc,Ra,Rb Rb,Ra,i03u			>>Rb[0:2] >>Rb[0:2] >>Rb[0:2] >>Rb[0:2] Ra (1) (2) (3) (4) → (1) (2) (3) (4) Rc		

Fig. 32

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 / 16
BS lsr system	SINGLE	Word	lsr	Rc,Ra,Rb Rb,Ra,i05u			Logical shift right >>>Rb[0:4] Ra → Rc	S1	32
		Pair word	lsrp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			>>>Ra[0:4] MHm MLm → MHm MLm Rb	S2	
	SIMD	Word	vlsr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u			MHm MLm → MHm MLm (1)>>>Ra[0:4] (2)>>>Ra[0:4] Rb		
		Half word	vlsrh	Rc,Ra,Rb Rb,Ra,i04u			>>>Rb[0:3] >>>Rb[0:3] Ra (1) (2) → (1) (2) Rc	S1	
		Byte	vlsrb	Rc,Ra,Rb Rb,Ra,i03u			>>>Rb[0:2] >>>Rb[0:2] Ra (1) (2) (3) (4) → (1) (2) (3) (4) Rc >>>Rb[0:2] >>>Rb[0:2]		
BS rotate system	SINGLE	Word	rol	Rc,Ra,Rb Rb,Ra,i05u			Ra → Rc <<Rb[4:0]	S1	32
	SIMD	Half word	vrolh	Rc,Ra,Rb Rb,Ra,i04u					
		Byte	vrolb	Rc,Ra,Rb Rb,Ra,i03u					
BS ext system	SINGLE	Word	extw	Mm,Rb,Ra			Ra → s··· MHm MLm Ra Rc	C	32
		Half word	exth	Ra2			Ra2 (1) → s··· (1) Ra2	S2	16
			exthu	Ra2			Ra2 (1) → 0··· (1) Ra2		
		Byte	extb	Ra2			Ra2 (1) → s··· (1) Ra2		
			extbu	Ra2			Ra2 (1) → 0··· (1) Ra2		
	SIMD	Half word	vexth	Mm,Rb,Ra			(1) (2) → s··· s··· MHm (1) MLm (2) Ra Rb	C	32

Fig. 33

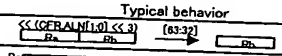
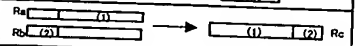
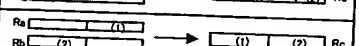
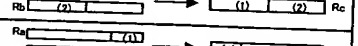
Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
CNV valn system	SIMD		valn	Rc,Ra,Rb	R:aln[1:0]			C	32
			valn1	Rc,Ra,Rb					
			valn2	Rc,Ra,Rb					
			valn3	Rc,Ra,Rb					
			valnvc1	Rc,Ra,Rb	R:VC0				
			valnvc2	Rc,Ra,Rb	R:VC0				
			valnvc3	Rc,Ra,Rb	R:VC0				
			valnvc4	Rc,Ra,Rb	R:VC0				

Fig. 34

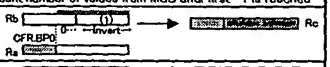
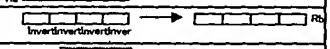
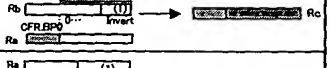
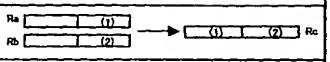
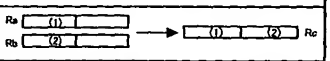
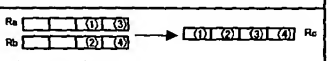
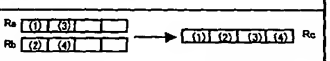
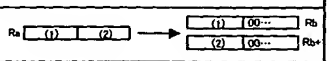
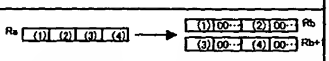
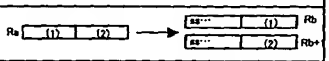
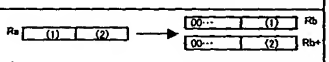
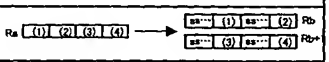
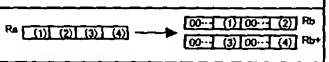
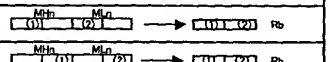
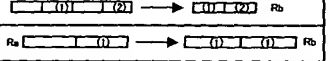
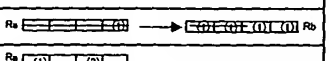
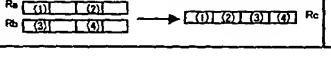
Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
CNV	SINGLE		bcnt1	Rb,Ra			Count the number of 1s	C	32
			bseq0	Rb,Ra			Count number of values from MSB until first 0 is reached		
			bseq1	Rb,Ra			Count number of values from MSB until first 1is reached		
			bseq	Rb,Ra			Count number of values from MSB until first −1 is reached		
			mskbrvh	Rc,Ra,Rb	R:BP0				
			byterev	Rb,Ra					
			mskbrvb	Rc,Ra,Rb	R:BP0				
	SIMD	Half word	vintlh	Rc,Ra,Rb					
			vintlhh	Rc,Ra,Rb					
		Byte	vintlb	Rc,Ra,Rb					
			vintlhb	Rc,Ra,Rb					
		Half word	vhunpkh	Rb:Rb+1,Ra					
		Byte	vhunpkb	Rb:Rb+1,Ra					
		Half word	vlunpkh	Rb:Rb+1,Ra					
			vlunpkhu	Rb:Rb+1,Ra					
		Byte	vlunpkb	Rb:Rb+1,Ra					
			vlunpkbu	Rb:Rb+1,Ra					
		Half word	vunpk1	Rb,Mn					
			vunpk2	Rb,Mn					
			vstovh	Rb,Ra					
		Byte	vstovb	Rb,Ra					
			vhpkb	Rc,Ra,Rb					

Fig. 35

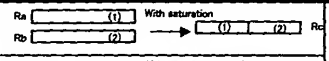
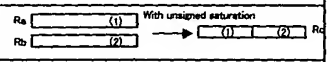
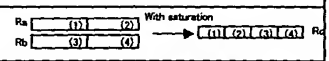
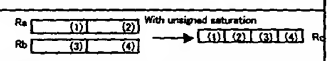
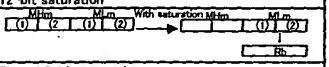
Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
SAT vlpk system	SIMD	Word-> Half word	vlpkh	Rc,Ra,Rb				C	32
			vlpkhu	Rc,Ra,Rb					
		Half word-> Byte	vlpkb	Rc,Ra,Rb					
			vlpkbu	Rc,Ra,Rb					
SAT sat system	SINGLE	Word	satw	Mm,Rb,Mn			Word saturation	C	32
			sath	Rb,Ra			Half wordsaturation		
			satb	Rb,Ra			Bytesaturation		
			satbu	Rb,Ra			Unsigned byte saturation		
			sat9	Rb,Ra			9-bit saturation		
			sat12	Rb,Ra			12-bit saturation		
	SIMD	Half word	vsath	Mm,Rb,Mn					
			vsath8	Rb,Ra			Signed 8-bit saturation		
			vsath8u	Rb,Ra			Unigned 8-bit saturation		
			vsath9	Rb,Ra			9-bit saturation		
			vsath12	Rb,Ra			12-bit saturation		

Fig. 36

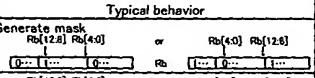
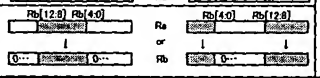
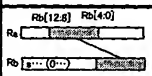
Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 / 16
MSK			mskgen	Rc,Rb Rb,I05U,i05u			Generate mask 	S2	32
			msk	Rc,Ra,Rb Rb,Ra,I05U,i05u					
EXTR			extr extru	Rc,Ra,Rb Rb,Ra,I05U,i05u Rc,Ra,Rb Rb,Ra,I05U,i05u			 With sign extension (Without sign extension)	S2	32
DIV			div divu	MHm,Rc,MHn,Ra,Rb MHm,Rc,MHn,Ra,Rb	W:ovs		Division	DIV	32
ETC			piNI			W:ih,ie,fie,pl R:PSR	Software interrupt N=0～7	B	32
			piN			W:ih,ie,pl R:PSR	Software interrupt N=0～7		16
			scN			W:ih,ie,pl R:PSR	System call N=0～7		
			ldstb	Rb,(Ra)			load bus lock	M	32
			rd	Rb,(Ra) Rb,(d11u)		R:eee	External register read		32
				Rb2,(Ra2)					16
			wt	(Ra),Rb (d11u),Rb		R:eee	External register write		32
				(Ra2),Rb2					16
			dpref	(Ra,d11u)			Pre-fetch		32
			dbgmN	i18u			N=0～3	DBGM	
			vcchk		W:CF R:VC		VC flag check	B	
			vmpsw vmpsw	 LR			VMP switching		
			vmpintd1 vmpintd2 vmpintd3			W:ie W:ie	VMP switching disabled		
			vmpinte1 vmpinte2 vmpinte3			W:ie	VMP switching enabled		
			nop				no operation	A	16

Fig. 37

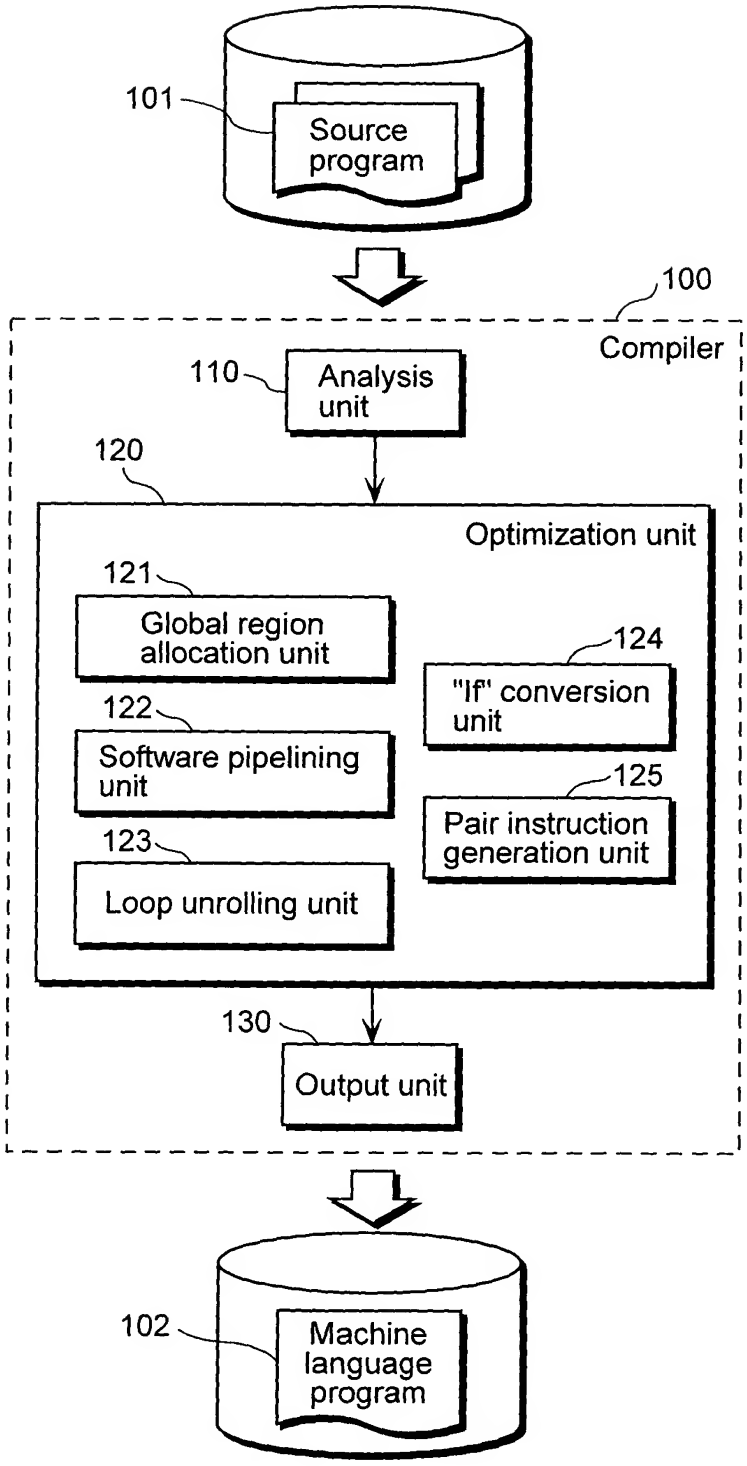

101
Source program
100
Compiler
110
Analysis unit
120
Optimization unit
121
Global region allocation unit
122
Software pipelining unit
123
Loop unrolling unit
124
"If" conversion unit
125
Pair instruction generation unit
130
Output unit
102
Machine language program

Fig. 38A

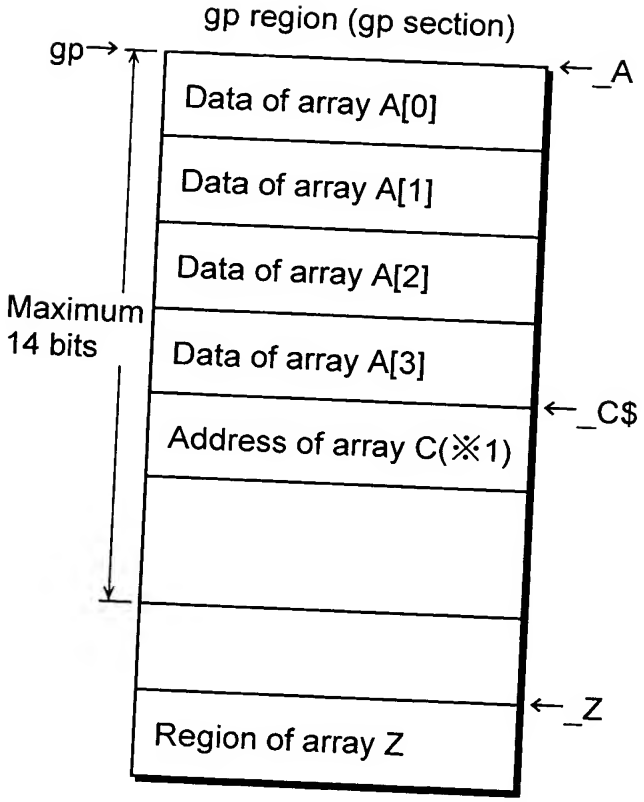


Fig. 38B

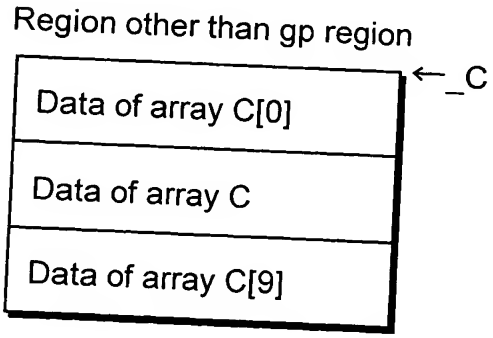


Fig. 39

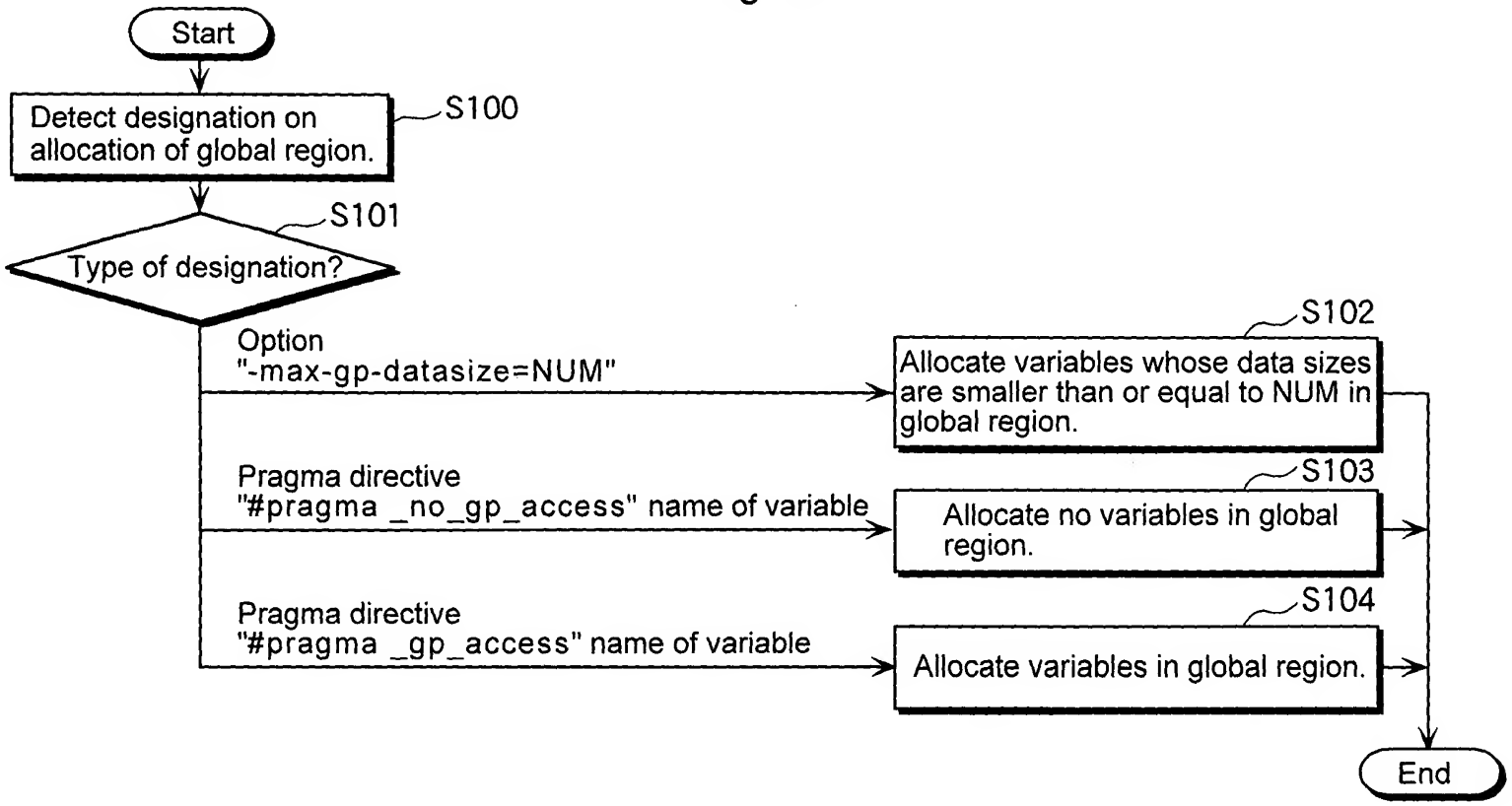

Start
Detect designation on allocation of global region.
S100
S101
Type of designation?
Option "-max-gp-datasize=NUM"
S102
Allocate variables whose data sizes are smaller than or equal to NUM in global region.
Pragma directive "#pragma _no_gp_access" name of variable
S103
Allocate no variables in global region.
Pragma directive "#pragma _gp_access" name of variable
S104
Allocate variables in global region.
End

Fig. 40

Comparison of generated codes by change of maximum data size

Default	Designate "-mmax-gp-datasize=40"

int a[8];
int c[10];
 	
int sample(void)
{
a[7] = c[0] + c[9];
 	
return a[7];
}
	int a[8];
	int c[10];
	
	int sample(void)
	{
	a[7] = c[0] + c[9];
	
	return a[7];
	}

ld r1,(gp,_c$ - .MN.gptop);;
setlo r0,LO(_c+36);;
sethi r0,HI(_c+36);;
ld r1,(r1);;
ld r0,(r0);;
add r0,r1,r0;;
st (gp,_a - .MN.gptop+28),r0
ret;;
	ld r1,(gp,_c - .MN.gptop);;
	ld r0,(gp,_c - .MN.gptop+36);;
	
	add r0,r1,r0;;
	st (gp,_a - .MN.gptop+28),r0
	ret;;

10 cycles 8 bytes	7 cycles 5 bytes

Fig. 41

When maximum data size to be allocated to gp region is 40

No size designation Defined outside of file	Defined inside of file/Size designation Defined outside of file

extern int a[ ];
extern int c[ ];
 	
int sample(void)
{
a[7] = c[0] + c[9];
 	
return a[7];
}
	int a[8];
	extern int c[10];
	
	int sample(void)
	{
	a[7] = c[0] + c[9];
	
	return a[7];
	}

setlo r0,LO(_c);;
setlo r1,LO(_c+36)
sethi r0,HI(_c);;
sethi r1,HI(_c+36);;
ld r3,(r0);;
setlo r2,LO(_a+28)
ld r0,(r1);;
sethi r2,HI(_a+28);;
add r1,r3,r0;;
mov r0,r1
st (r2),r1
ret ;;
	ld r1,(gp,_c - .MN.gptop);;
	
	ld r0,(gp,_c - .MN.gptop+36);;
	
	add r0,r1,r0;;
	
	st (gp,_a - .MN.gptop+28),r0
	ret;;

10 cycles 12 bytes	7 cycles 5 bytes

Fig. 42

When maximum data size to be allocated to gp region is default of 32

Size designation Defined outside of file/ Defined inside of file "#pragma _no_gp_access" directive	No size designation Defined outside of file/ Defined inside of file "#pragma _gp_access" directive

#pragma _no_gp_access a, c
 	
extern int a[8];
int c[10];
 	
int sample(void)
{
a[7] = c[0] + c[9];
 	
return a[7];
}
	#pragma _gp_access a, c
	
	extern int a[];
	int c[10];
	
	int sample(void)
	{
	a[7] = c[0] + c[9];
	
	return a[7];
	}

setlo r0,LO(_c);;
setlo r1,LO(_c+36)
sethi r0,HI(_c);;
sethi r1,HI(_c+36);;
ld r3,(r0);;
setlo r2,LO(_a+28)
ld r0,(r1);;
sethi r2,HI(_a+28);;
add r1,r3,r0;;
mov r0,r1
st (r2),r1
ret ;;
	ld r1,(gp,_c - .MN.gptop);;
	
	ld r0,(gp,_c - .MN.gptop+36);;
	
	add r0,r1,r0;;
	
	st (gp,_a - .MN.gptop+28),r0
	ret;;

10 cycles 12 bytes	7 cycles 5 bytes

Fig. 43A

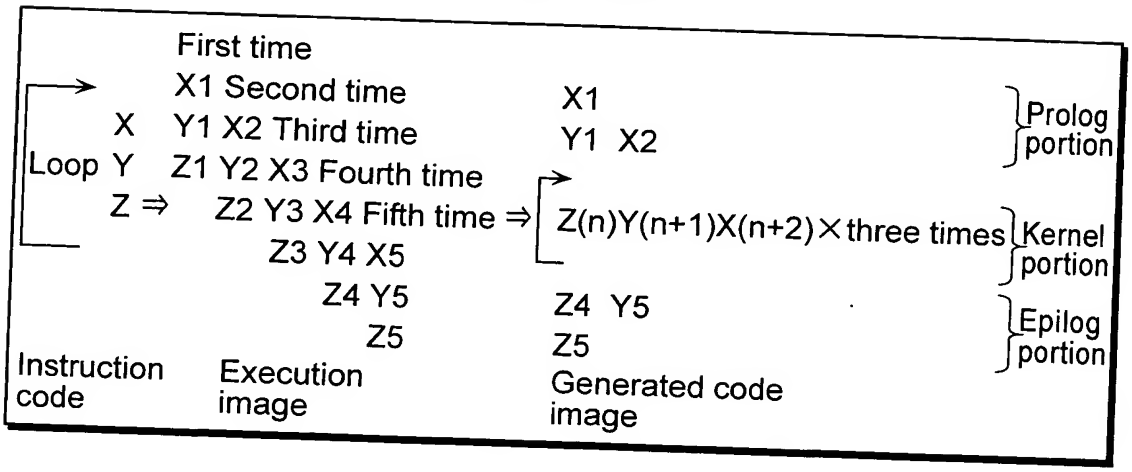


Fig. 43B

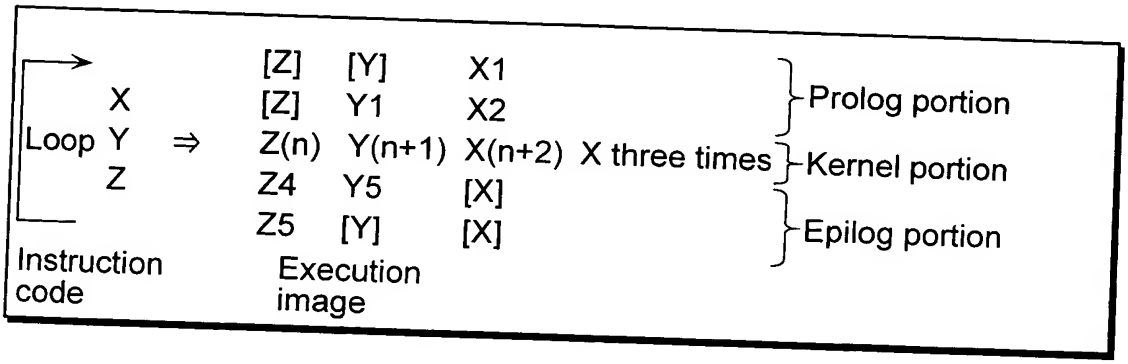


Fig. 43C

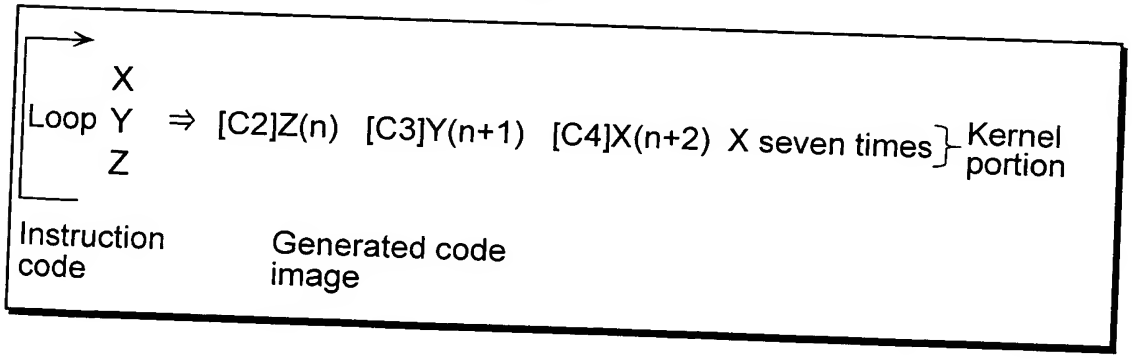


Fig. 44

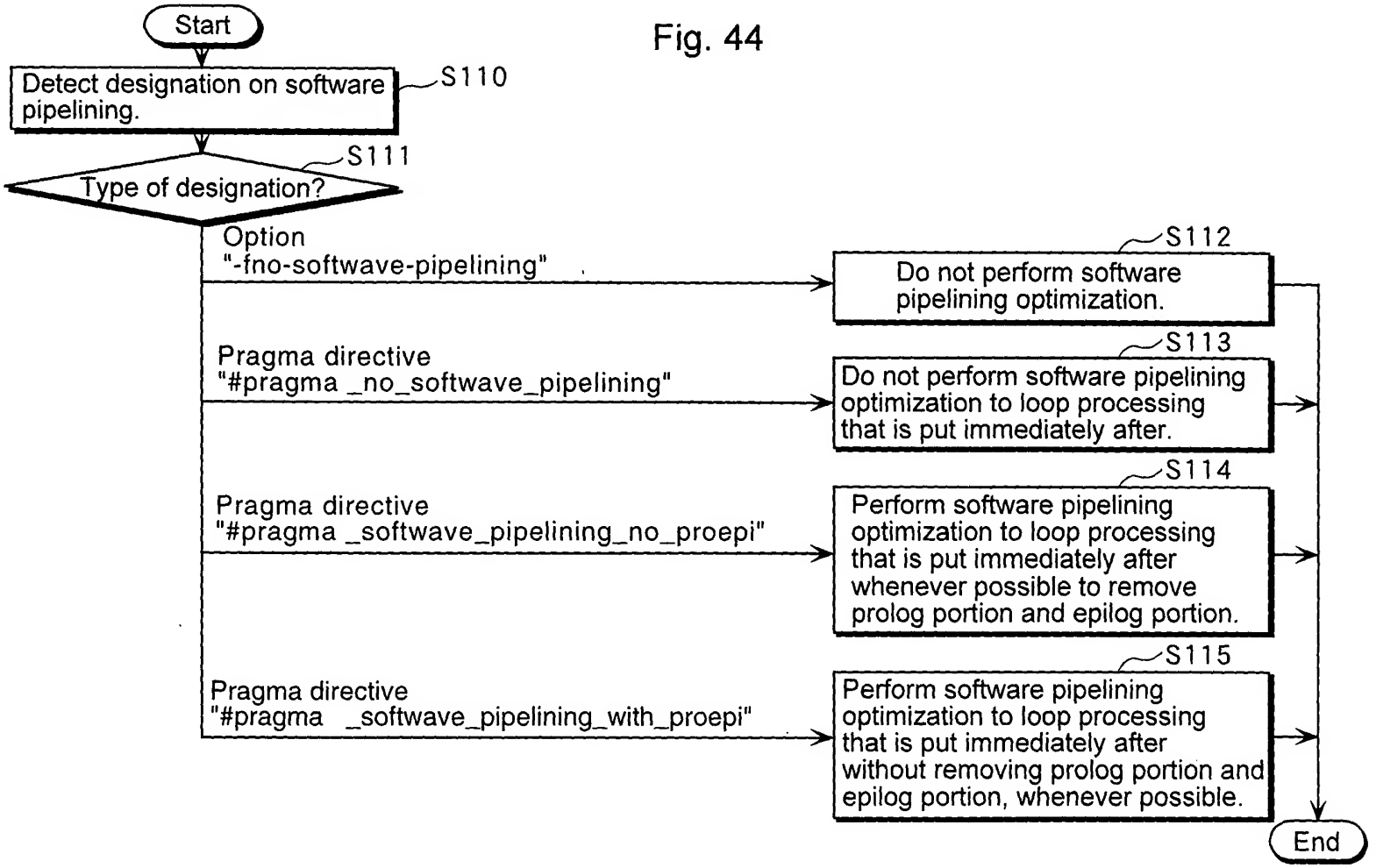

Start
Detect designation on software pipelining.
S110
Type of designation?
S111
Option "-fno-softwave-pipelining"
S112
Do not perform software pipelining optimization.
Pragma directive "#pragma _no_softwave_pipelining"
S113
Do not perform software pipelining optimization to loop processing that is put immediately after.
Pragma directive "#pragma _softwave_pipelining_no_proepi"
S114
Perform software pipelining optimization to loop processing that is put immediately after whenever possible to remove prolog portion and epilog portion.
Pragma directive "#pragma _softwave_pipelining_with_proepi"
S115
Perform software pipelining optimization to loop processing that is put immediately after without removing prolog portion and epilog portion, whenever possible.
End

Fig. 45

Compilation with compile option -0

Default	"#pragma _software_pipelining_with_proepi" directive

int a[100];
int b[100];
void sample(int c)
{
int i;
 	
for(i=0; i<100; i++) {
a[i] = b[i] + c;
}
}
	int a[100];
	int b[100];
	void sample(int c)
	{
	int i;
	#pragma _software_pipelining_with_proepi
	for(i=0; i<100; i++) {
	a[i] = b[i] + c;
	}
	}

ld r5,(gp,_x$ - .MN.gptop);;
 	
mov r1,98
settar C6,C4,L00030
ld r4,(gp,_v$ - .MN.gptop);;
L00030
[C4] add r2,r3,r0
[C6] ld r3,(r5+);;
[C4] st (r4+),r2
[C6] jloop C6,C4,tar,r1,-1;;
 	
ret ;;
	ld r5,(gp,_x$ - .MN.gptop);;
	ld r4,(r5+);;
	mov r1,96
	settar C6,L00023
	ld r3,(gp,_v$ - .MN.gptop);;
	L00023
	add r2,r4,r0
	ld r4,(r5+);;
	st (r3+),r2
	[C6] jloop C6,C4,tar,r1,-1;;
	add r2,r4,r0;;
	st (r3+),r2
	ret ;;

2 + 2×101 + 3 = 207 cycles 9 bytes	3+ 2×99 + 3 = 204 cycles 12 bytes

Fig. 46

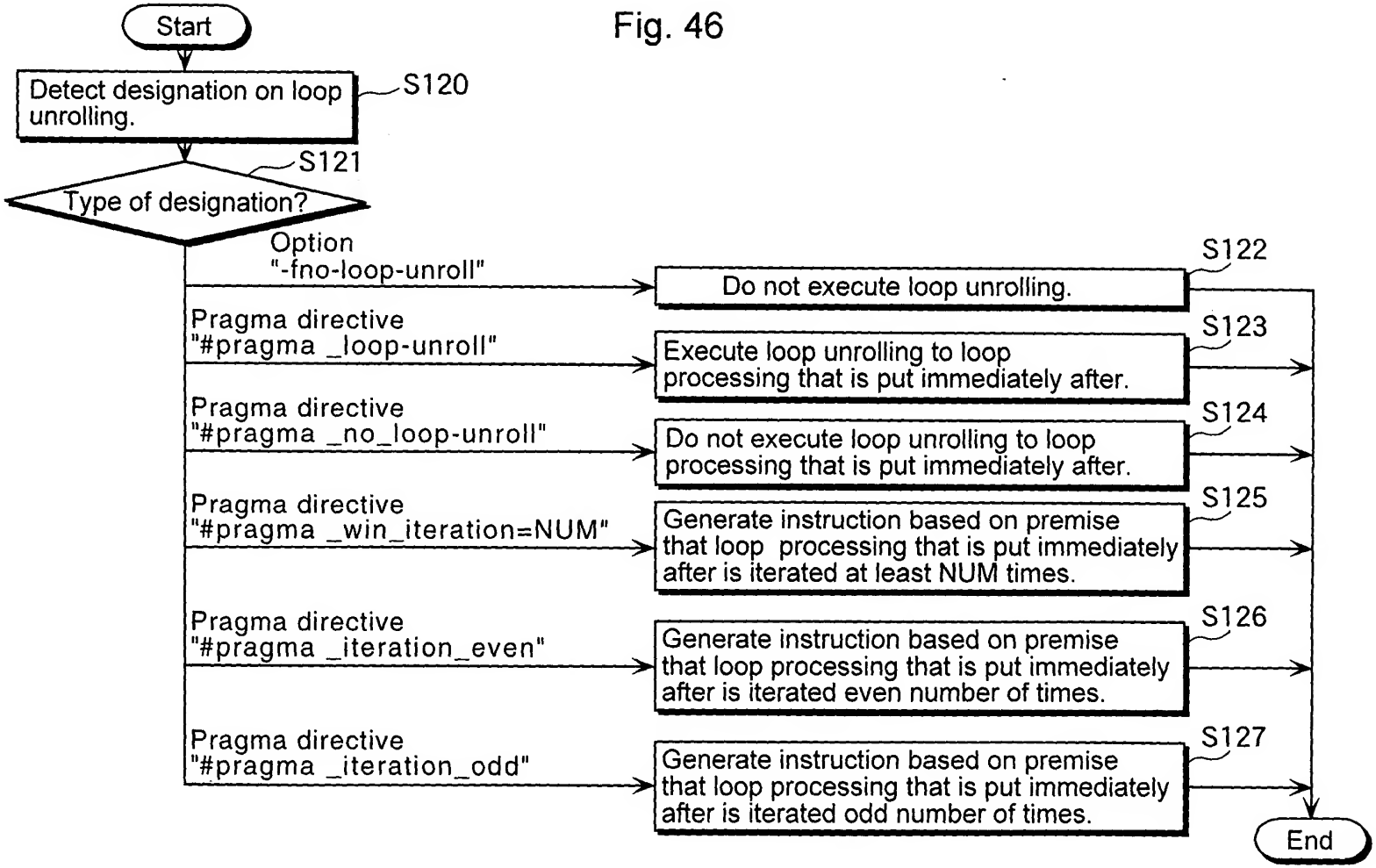

Start
Detect designation on loop unrolling.
S120
S121
Type of designation?
Option "-fno-loop-unroll"
Do not execute loop unrolling.
S122
Pragma directive "#pragma _loop-unroll"
Execute loop unrolling to loop processing that is put immediately after.
S123
Pragma directive "#pragma _no_loop-unroll"
Do not execute loop unrolling to loop processing that is put immediately after.
S124
Pragma directive "#pragma _win_iteration=NUM"
Generate instruction based on premise that loop processing that is put immediately after is iterated at least NUM times.
S125
Pragma directive "#pragma _iteration_even"
Generate instruction based on premise that loop processing that is put immediately after is iterated even number of times.
S126
Pragma directive "#pragma _iteration_odd"
Generate instruction based on premise that loop processing that is put immediately after is iterated odd number of times.
S127
End

Fig. 47

Example of increased execution speed by "#pragma _loop_unroll" directive

Default	"#pragma _loop_unroll" directive

int a[100];
int b[100];
void sample(int c)
{
int i, ret=0;
 	
for(i=0; i<100; i++) {
ret += a[i] * a[i];
}
return ret;
}
	int a[100];
	int b[100];
	void sample(int c)
	{
	int i, ret=0;
	#pragma _loop_unroll
	for(i=0; i<100; i++) {
	ret += a[i] * a[i];
	}
	return ret;
	}

mov r0,0
mov r1,98;;
settar C6,C4,L00013
 	
ld r3,(gp,_a$ - .MN.gptop)
mul m0,r2,gp,0;;
L00013
[C4] mac m0,r0,r2,r2,m0
 	
[C6] ld r2,(r3+)
 	
[C6] jloop C6,C4,tar,r1,-1;;
ret ;;
	ld r3,(gp,_a$ - .MN.gptop);;
	mov r0,0
	mov r1,48;;
	settar C6,C2:C4,L00013
	add r5,r3,4
	
	mul m0,r2,gp,0;;
	L00013
	[C2] mac m0,r0,r2,r2,m0
	[C3] add r3,r3,8
	[C3] ld r2,(r5+)8;;
	[C3] mac m0,r0,r4,r4,m0
	[C4] ld r4,(r3)
	[C6] jloop C6,C2:C4,tar,r1,-1;;
	ret ;;

2 + 2×101 + 3 = 207 cycles 9 bytes	3 + 2× 52 + 3 = 110 cycles 13 bytes

Fig. 48

Default	"#pragma _loop_unroll" directive and "#pragma _align_local_pointer" directive

int a[100];
int b[100];
void sample(int c)
{
int i;
 	
int *pa=a;
int *pb=b;
 	
for(i=0; i<100; i++) {
*pa++ = i * c >> *pb++;
}
}
	int a[100];
	int b[100];
	void sample(int c)
	{
	int i;
	#pragma _align_local_pointer=8 pa,pb
	int *pa=a;
	int *pb=b;
	#pragma _loop_unroll
	for(i=0; i<100; i++) {
	*pa++ = i * c >> *pb++;
	}
	}

mov r4,0
ld r6,(gp,_a$ - .MN.gptop);;
mov r1,98
settar C6,C4,L00025
ld r5,(gp,_b$ - .MN.gptop);;
L00025
[C4] asr r2,r4,r3;;
 	
[C6] ld r3,(r5+);;
[C4] add r4,r4,r0
[C4] st (r6+),r2
[C6] jloop C6,C4,tar,r1,-1;;
ret ;;
	mov r6,0
	ld r8,(gp,_a$ - .MN.gptop);;
	mov r1,48
	settar C6,C4,L00016
	ld r7,(gp,_b$ - .MN.gptop);;
	L00016
	[C2] add r6,r6,r0;;
	[C2] stp (r8+),r2:r3
	[C3] asr r2,r6,r4
	[C3] add r6,r6,r0;;
	[C3] asr r3,r6,r5
	[C4] ldp r4:r5,(r7+)
	[C6] jloop C6,C2:C4,tar,r1,-1;;
	ret ;;

2 + 3×101 + 3 = 308 cycles 11 bytes	2 + 3× 51 + 3 = 158 cycles 13 bytes

Fig. 49

Without "#pragma _min_iteration" directive	With "#pragma _min_iteration" directive

int a[101];
int b[101];
void sample(int c, int end)
{
int i;
int *pa=a;
int *pb=b;
 	
for (i = 0; i <end; i++) {
*pa++ = i * c >> *pb++;
}
*pa = end;
}
	int a[101];
	int b[101];
	void sample(int c, int end)
	{
	int i;
	int *pa=a;
	int *pb=b;
	#pragma _min_iteration=4
	for (i = 0; i <end; i++) {
	*pa++ = i * c >> *pb++;
	}
	*pa = end;
	}

cmple C0,r1,0
ld r5,(gp,_a$ - .MN.gptop);;
mov r4,0
ld r6,(gp,_b$ - .MN.gptop)
[C0] br L00016;;
mov r3,0
settar L00017
L00017
ld r2,(r6+);;
add r4,1;;
cmplt C0,r4,r1
asr r2,r3,r2;;
add r3,r3,r0
st (r5+),r2
[C0] jmpf tar;;
L00016
st (r5),r1
ret ;;
	mov r5,0
	ld r7,(gp,_a$ - .MN.gptop);;
	settar C6,C4,L00027
	sub r2,r1,2
	ld r6,(gp,_b$ - .MN.gptop);;
	
	L00027
	[C4] asr r3,r5,r4;;
	[C6] ld r4,(r6+);;
	[C4] add r5,r5,r0
	[C4] st (r7+),r3
	[C6] jloop C6,C4,tar,r2,-1;;
	
	st (r7),r1
	ret ;;

2 + 4×100 + 3 = 405 cycles 16 bytes	2 + 3× 101 + 3 = 308 cycles 12 bytes

Fig. 50

Example of loop unrolling being impossible

C language source with unknown number of loops	Result of compilation

void sample(int c, int end)
{
int i;
#pragma _align_local_pointer=8 pa, pb
int *pa=a;
int *pb=b;
#pragma _loop_unroll
#pragma _min_iteration=4
for (i = 0; i <end; i++) {
*pa++ = i * c >> *pb++;
}
*pa = end;
}
	mov r5,0
	ld r7,(gp,_a$ - .MN.gptop);;
	settar C6,C4,L00026
	sub r2,r1,2
	ld r6,(gp,_b$ - .MN.gptop);;
	L00026
	[C4] asr r3,r5,r4;;
	[C6] ld r4,(r6+);;
	[C4] add r5,r5,r0
	[C4] st (r7+),r3
	[C6] jloop C6,C4,tar,r2,-1;;
	st (r7),r1
	ret ;;

Fig. 51

"#pragma _iteration_even" directive	"#pragma _iteration_odd" directive

int a[101];
int b[101];
void sample(int c, int end)
{
int i;
#pragma _align_local_pointer=8 pa, pb
int *pa=a;
int *pb=b;
#pragma _min_iteration=50
#pragma _loop_unroll
#pragma _iteration_even
for (i = 0; i <end; i++) {
*pa++ = i * c >> *pb++;
}
*pa = end;
}
	int a[101];
	int b[101];
	void sample(int c, int end)
	{
	int i;
	#pragma _align_local_pointer=8 pa, pb
	int *pa=a;
	int *pb=b;
	#pragma _min_iteration=50
	#pragma _loop_unroll
	#pragma _iteration_odd
	for (i = 0; i <end; i++) {
	*pa++ = i * c >> *pb++;
	}
	*pa = end;
	}

sub r2,r1,6;;
mov r6,0
asr r2,1
ld r7,(gp,_a$ - .MN.gptop);;
settar C6,C4,L00036
add r2,1
ld r8,(gp,_b$ - .MN.gptop);;
L00036
[C4] asr r3,r6,r4;;
[C4] add r6,r6,r0
[C4] st (r7+),r3;;
[C4] asr r3,r6,r5;;
[C6] ldp r4:r5,(r8+);;
[C4] add r6,r6,r0
[C4] st (r7+),r3
[C6] jloop C6,C4,tar,r2,-1;;
 	
st (r7),r1
ret ;;
	sub r2,r1,7;;
	mov r6,0
	asr r2,1
	ld r7,(gp,_a$ - .MN.gptop);;
	settar C6,C4,L00046
	add r2,1
	ld r8,(gp,_b$ - .MN.gptop);;
	L00046
	[C4] asr r3,r6,r4;;
	[C4] add r6,r6,r0
	[C4] st (r7+),r3;;
	[C4] asr r3,r6,r5;;
	[C6] ldp r4:r5,(r8+);;
	[C4] add r6,r6,r0
	[C4] st (r7+),r3
	[C6] jloop C6,C4,tar,r2,-1;;
	ld r2,(r8+);;
	asr r2,r6,r2;;
	st (r7+),r2;;
	st (r7),r1
	ret ;;

Fig. 52

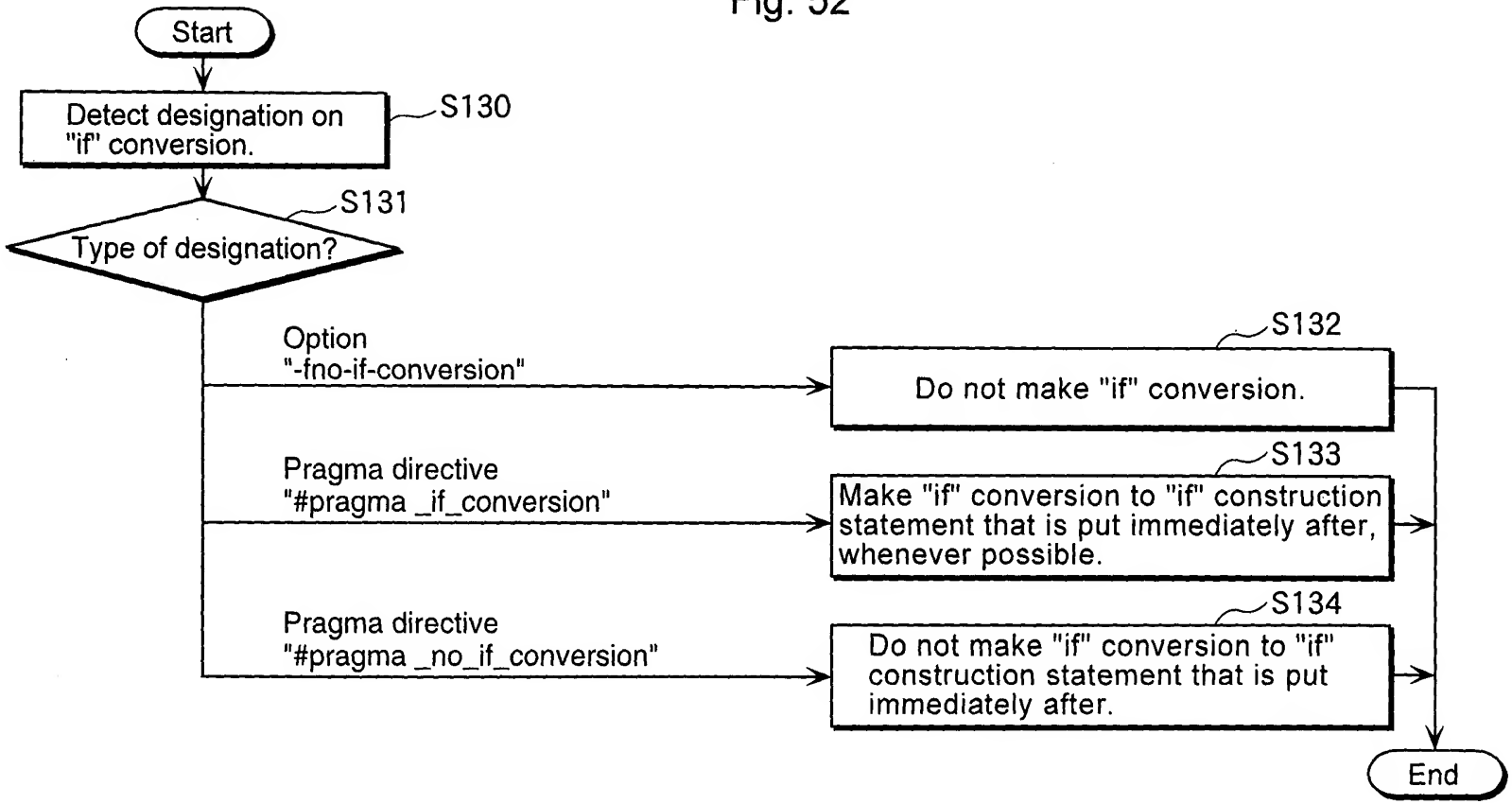

Start
Detect designation on "if" conversion.
S130
S131
Type of designation?
Option "-fno-if-conversion"
S132
Do not make "if" conversion.
Pragma directive "#pragma _if_conversion"
S133
Make "if" conversion to "if" construction statement that is put immediately after, whenever possible.
Pragma directive "#pragma _no_if_conversion"
S134
Do not make "if" conversion to "if" construction statement that is put immediately after.
End

Fig. 53

"#pragma _no_if_conversion" directive	"#pragma _if_conversion" directive

int func(int a)
{
  int b ;
  #pragma _no_if_conversion
  if(a == 2)
  b = 0 ;
  else
  b = 1 ;
  return b ;
}
	int func(int a)
	{
	  int b ;
	  #pragma _if_conversion
	  if(a == 2)
	  b = 0 ;
	  else
	  b = 1 ;
	  return b ;
	}

cmpne C0,r0,2 ;;
[C0] br L00002 ;; // Branch is generated.
mov r0,0
ret ;;
L00002
mov r0,1
ret ;; ;;
	cmpeq C0:C1,r0,2 ;;
	[C0] mov r0,0 // Execution instruction with condition is
	[C1] mov r0,1 // generated instead of branch instruction.
	ret ;;

"then" side: 5 cycles, "else" side 7 cycles 12 bytes	4 cycles 8 bytes

Fig. 54

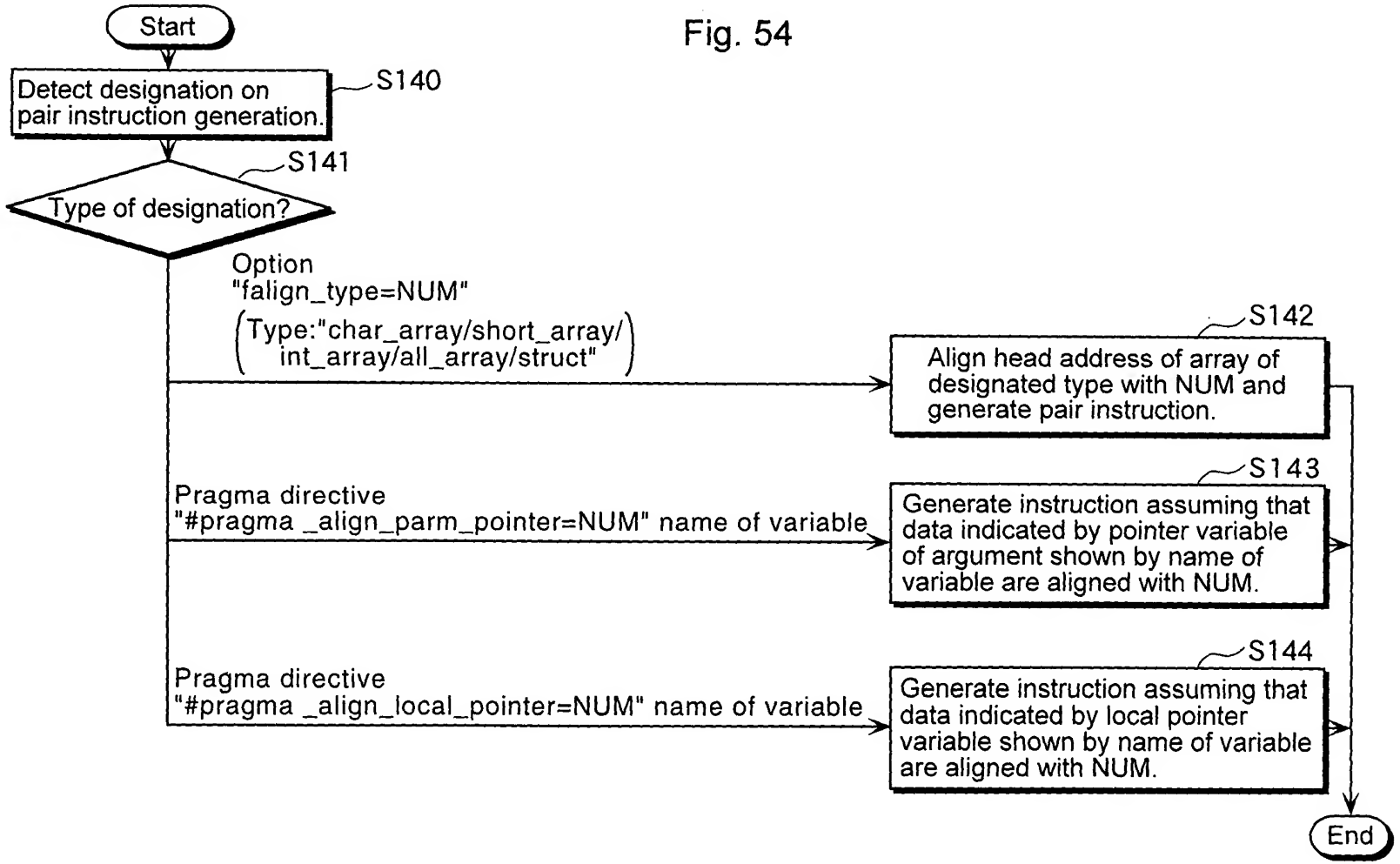

Start
Detect designation on pair instruction generation.
S140
S141
Type of designation?
Option "falign_type=NUM" (Type:"char_array/short_array/int_array/all_array/struct")
S142
Align head address of array of designated type with NUM and generate pair instruction.
Pragma directive "#pragma _align_parm_pointer=NUM" name of variable
S143
Generate instruction assuming that data indicated by pointer variable of argument shown by name of variable are aligned with NUM.
Pragma directive "#pragma _align_local_pointer=NUM" name of variable
S144
Generate instruction assuming that data indicated by local pointer variable shown by name of variable are aligned with NUM.
End

Fig. 55

No designation of alignment	Designation of option "-falign_short_array=4"

short src[8] ;
short dst[8] ;
 	
void align1(void)
{
char dummy ;
int i ;
 	
for(i=0 ; i<8 ; i+=2)
dst[i] = src[i] + src[i+1] ;
}
	No change

addu r5,gp,_src - .MN.gptop
addu r4,gp,_src - .MN.gptop+2 ;;
addu r3,gp,_dst - .MN.gptop
mov r0,2
settar C6,C4,L00005 ;;
L00005
[C4] ldh r1,(r4+)4 ;; //As alignment is unknown,
[C4] add r1,r2,r1 //two data are loaded
[C6] ldh r2,(r5+)4 ;; //independently (ldh instruction).
[C4] sth (r3+)4,r1
[C6] jloop C6,C4,tar,r0,-1 ;; //L00005
ret ;;
	addu r5,gp,_src - .MN.gptop ;;
	addu r4,gp,_dst - .MN.gptop
	mov r0,2
	settar C6,C4,L00005 ;;
	L00005
	[C4] add r1,r2,r3 // As alignment is 4,
	[C6] ldhp r2:r3,(r5+) ;; // pair access instruction
	[C4] sth (r4+)4,r1 // (ldhp)is generated.
	[C6] jloop C6,C4,tar,r0,-1 ;; //L00005
	ret ;;

2+4×5+3=25 cycles 22 bytes	2+2×5+3=15 cycles 18 bytes

Fig. 56

No designation of alignment	Use "#pragma _align_ parm_pointer" directive

int align2(short * src)
{
int v = 0 ;
for(int i=0 ; i<100 ; i+=2){
v += src[i] * src[i+1] ;
}
return v ;
}
	#pragma _align_parm_pointer=4 src
	int align2(short * src)
	{
	int v = 0 ;
	for(int i=0 ; i<100 ; i+=2){
	v += src[i] * src[i+1] ;
	}
	return v ;
	}

mov r5,0
mov r1,48
settar C6,C4,L00005 ;;
mov r4,r0
add r3,r0,2
mul m0,r2,gp,0 ;;
L00005
[C4] ldh r0,(r3+)4 ;; // Apply software pipelining.
[C4] lmac m0,r5,r2,r0,m0 // As alignment of data is unknown,
[C6] ldh r2,(r4+)4 // execute ldh instruction twice.
[C6] jloop C6,C4,tar,r1,-1 ;; //L00005
mov r0,r5
ret ;;
	mov r4,0 ;;
	mov r1,48
	settar C6,C4,L00005
	mul m0,r2,gp,0 ;;
	L00005
	[C4] lmac m0,r4,r2,r3,m0 // Apply software pipelining.
	[C6] ldhp r2:r3,(r0+) // Pair access instruction is usable.
	[C6] jloop C6,C4,tar,r1,-1 ;; //L00005
	mov r0,r4
	ret ;;

2+3×51+3=160 cycles 24 bytes	2+2×51+3=107 cycles 18 bytes

Fig. 57

No designation of alignment	Use "#pragma _align_local_ pointer" directive

void align3(int n)
{
short * from ;
short * to ;
int i ;
 	
from = &(src[n]) ; to = &(dst[n]) ;
for(i=0 ; i<16 ; i++, from+=2, to+=2){
* to = * from ;
* (to+1) = * (from+1) ;
}
}
	void align3(int n)
	{
	# pragma _align_local_pointer=4 from,to
	short * from ;
	short * to ;
	int i ;
	
	from = &(src[n]) ; to = &(dst[n]) ;
	for(i=0 ; i<16 ; i++, from+=2, to+=2){
	* to =* from ;
	* (to+1) = * (from+1) ;
	}
	}

ld r3,(gp,_src$ - .MN.gptop) ;;
ld r2,(gp,_dst$ - .MN.gptop) ;;
add r1,r0,r0 ;;
mov r0,13
add r6,r3,r1
add r5,r2,r1 ;;
settarC6,L00016
add r4,r5,2
add r3,r6,2 ;;
L00016
ldh r1,(r6+)4 ;;
ldh r2,(r3+)4 ;;
sth (r5+)4,r1 ;;
sth (r4+)4,r2
[C6] jloop C6,tar,r0,-1 ;; //L00016
ret ;;
	ld r3,(gp,_src$ - .MN.gptop) ;;
	ld r2,(gp,_dst$ - .MN.gptop) ;;
	mov r1,13
	add r0,r0,r0 ;;
	settarC6,L00016
	add r4,r3,r0
	add r0,r2,r0 ;;
	L00016
	ldhp r2:r3,(r4+) ;;
	sthp (r0+),r2:r3
	[C6] jloop C6,tar,r1,-1 ;; //L00016
	ret ;;

5+4×16+3=72 cycles 30 bytes	5+3×16+3=56 cycles 22 bytes